杭州艾研信息技术有限公司

# 树莓派上的物联网实现

讲述物联网（Internet Of Things）在树莓派（Raspberry 3）上通过 WebIOPI 的实现，可以通过网络查找、访问、监控多种传感器终端

# 1. 背景介绍

## 1.1. 树莓派（Raspberry）

### 1.1.1. 简介

图 1-1 Raspberry 3 示意图

Raspberry Pi(中文名为“树莓派”,简写为 RPi，(或者 RasPi / RPI)[1] 是为学习计算机编程教育而设计，只有信用卡大小的微型电脑，其系统基于 Linux。随着 Windows 10 IoT 的发布,我们也将可以用上运行 Windows 的树莓派。自问世以来，受众多计算机发烧友和创客的追捧，曾经一“派”难求。别看其外表“娇小”，内“心”却很强大，视频、音频等功能通通皆有，可谓是“麻雀虽小，五脏俱全”。

它是一款基于 ARM 的微型电脑主板，以 SD/MicroSD 卡为内存硬盘，卡片主板周围有 1/2/4 个 USB 接口和一个 10/100 以太网接口（A 型没有网口），可连接键盘、鼠标和网线，同时拥有视频模拟信号的电视输出接口和 HDMI 高清视频输出接口，以上部件全部整合在一张仅比信用卡稍大的主板上，具备所有 PC 的基本功能只需接通电视机和键盘，就能执行如电子表格、文字处理、玩游戏、播放高清视频等诸多功能。 Raspberry Pi B 款只提供电脑板，无内存、电源、键盘、

机箱或连线。

但是 2016 年出的新款 Raspberry 3 却实实在在的做了全方位的升级！从树莓派 3 的第一印象来看，外观和树莓派 2 几乎没啥大的差别，板载资源也“基本相同”。但是仔细一看的话，我们还是能看到器件以及电路的不少改动，对比树莓派 2，我们来看看主要有哪些改动？

- 在 CPU 上做了全方位的升级，从 32 位 A7（BCM2836）升级到 64 位 A53（BCM2837），主频从 900MHz 升级到 1.2GHz；
- GPU 核心没变，但是主频从 250MHz 提升到 400MHz
- 板卡背面增加了 WiFi/BLE 电路，方便对智能产品的开发；
- 供电电路升级到 2.5A@5V，增加了扩展更多模块的可能性；
- 两颗指示灯也因天线的布局移到了电源一侧；

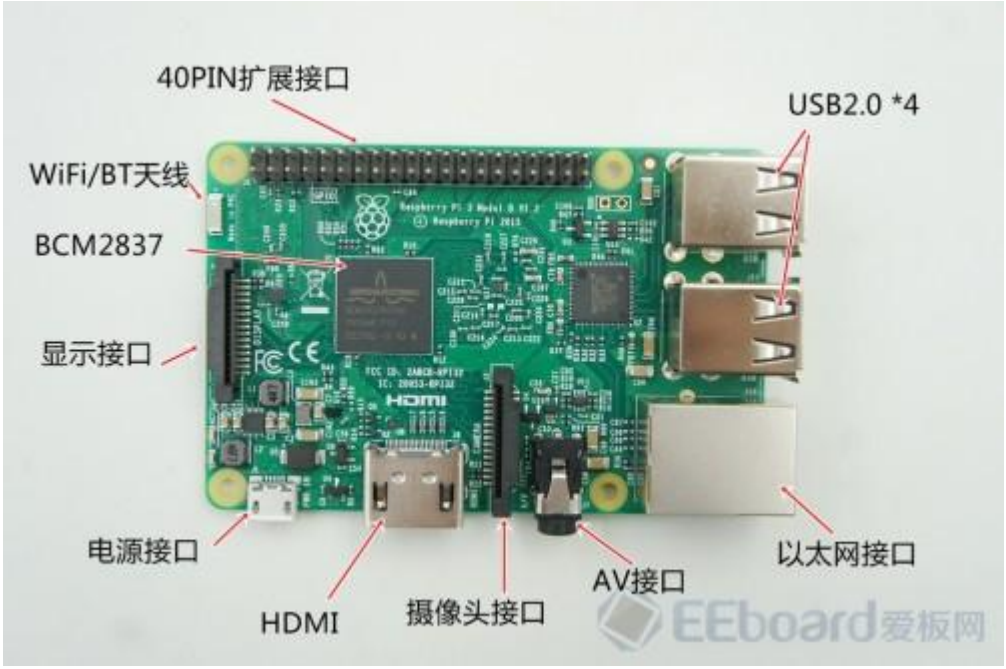



图 1-2 Raspberry 3 正面


图 1-3 Raspberry 3 反面

除了以上这些大的改动外，树莓派 3 也有细小的改动之处，比如 MicroSD 卡座变得更加小巧，不再是那种弹扣式结构，可以直接插拔的。目前树莓派 3 本身的资料不是太多，加之采用博通这个公司的应用处理器，资料很难完全的开源，我们也难很形象的去描述这颗应用处理器的性能，不妨实际上电测试一下，当然作为实测，需要有个对比对象，这时，树莓派 2 当然当仁不让的承担这个任务，充当下“绿叶”。

系统是使用树莓派官方最新的 raspbian-jessie，通过 win32disk 烧写工具烧录在 MicroSD 卡，插上相应的显示器、键盘、鼠标等，上电启动即可。

## 1.2. 物联网（Internet Of Things）

### 1.2.1. 简介

物联网是新一代信息技术的重要组成部分，也是“信息化”时代的重要发展阶段。其英文名称是：“Internet of things（IoT）”。顾名思义，物联网就是物物相连的互联网。这有两层意思：其一，物联网的核心和基础仍然是互联网，是在互联网基础上的延伸和扩展的网络；其二，其用户端延伸和扩展到了任何物品与物品之间，进行信息交换和通信，也就是物物相息。物联网通过智能感知、识别技术与普适计算等通信感知技术，广泛应用于网络的融合中，也因此被称为继计算机、互联网之后世界信息产业发展的第三次浪潮。物联网是互联网的应用拓展，

与其说物联网是网络，不如说物联网是业务和应用。因此，应用创新是物联网发展的核心，以用户体验为核心的创新 2.0 是物联网发展的灵魂。

把所有物品通过信息传感设备与互联网连接起来，进行信息交换，即物物相息，以实现智能化识别和管理。这是物联网“一句式”理解。

## 1.2.2. 应用领域

这几年推行的智能家居其实就是把家中的电器通过网络控制起来。可以想见，物联网发展到一定阶段，家中的电器可以和外网连接起来，通过传感器传达电器的信号。厂家在厂里就可以知道你家中电器的使用情况，也许在你之前就知道你家电器的故障。某一天突然有维修工上门告诉你家中空调有问题，你还惊异地不相信。物联网的发展，必然带动传感器的发展，传感器发展到一定程度，变形金刚会真地出现在我们的面前。物联网在最近几年的发展中已经渐渐的积淀深入到和很多分支领域中区了。

### 1.2.2.1. 智能家居

智能家居是利用先进的计算机技术，运用智能硬件（氦氪 wifi、Zigbee、蓝牙、NB-iot 等），物联网技术，通讯技术，将与家具生活的各种子系统有机的结合起来，通过统筹管理，让家居生活更舒适，方便，有效，与安全。


图 1-4 智能家居示意图

### 1.2.2.2. 智慧交通

智慧交通，是将物联网、互联网、云计算为代表的智能传感技术、信息网络技术、通信传输技术和数据处理技术等有效地集成，并应用到整个交通系统中，

在更大的时空范围内发挥作用的综合交通体系[3] 。智慧交通是以智慧路网、智慧出行、智慧装备、智慧物流、智慧管理为重要内容，以信息技术高度集成、信息资源综合运用为主要特征的大交通发展新模式。依托迪蒙科技在云计算、物联网、大数据、金融科技等领域的丰富开发经验和雄厚的技术积累，历时 3 年倾力打造的中国目前首家 一款集网约专车、智慧停车、汽车租赁、汽车金融，以及其他智慧出行领域创新商业模式于一体的高端智慧交通整体解决方案。

#### 1.2.2.3. 智能电网

智能电网是在传统电网的基础上构建起来的集传感、通信、计算、决策与控制为一体的综合数物复合系统，通过获取电网各层节点资源和设备的运行状态，进行分层次的控制管理和电力调配，实现能量流、信息流和业务流的高度一体化，提高电力系统运行稳定性，以达到最大限度地提高设备效利用率，提高安全可靠性，节能减排，提高用户供电质量，提高可再生能源的利用效率。

#### 1.2.2.4. 智慧城市

智慧城市就是运用信息和通信技术手段感测、分析、整合城市运行核心系统的各项关键信息，从而对包括民生、环保、公共安全、城市服务、工商业活动在内的各种需求做出智能响应。其实质是利用先进的信息技术，实现城市智慧式管理和运行，进而为城市中的人创造更美好的生活，促进城市的和谐、可持续成长。

随着人类社会的不断发展，未来城市将承载越来越多的人口。目前，我国正处于城镇化加速发展的时期，部分地区“城市病”问题日益严峻。为解决城市发展难题，实现城市可持续发展，建设智慧城市已成为当今世界城市发展不可逆转的历史潮流。

智慧城市的建设在国内外许多地区已经展开，并取得了一系列成果，国内的如智慧上海、智慧双流；国外如新加坡的“智慧国计划”、韩国的“U-City 计划”等

## 1.3. 基于树莓派的物联网的优势

如今距离 2012 年 2 月第一代树莓派诞生已经过去了四年，现在全新的树莓派 3 也已经来到了我们的身边。树莓派 3 采用了 64 位四核 ARM Cortex 1.2GHz 处理器，具有内置 WiFi 与蓝牙。使这款小型设备更为实用的并不是它更快的处理器，而是它的网络支持。树莓派 3 或许将成为许多物联网项目的硬件平台选择。物联网将对我们家中与工作环境中的事物进行改造，使它们能够自动与其他设备

进行交流。

例如可穿戴的血压监控器将与房间中的恒温器与暖气阀门想沟通，又或者能联系当地的医生寻求建议。而在工作场所中，每个设备将具备联网能力，传感器也将能处理数据。最新的树莓派的到来使这一切距离现实更近一步。

### 1.3.1. 性能、联网与便利性

小型的单块板状的计算机往往都有所限制，而树莓派则通过结合子卡的形式来添加一些输入输出能力。例如树莓派与 Z-Wave 这款智能家居设备的无线沟通技术相结合。通过这种方式树莓派可以被应用到很多场景之中。

树莓派 3 在尺寸上仅有一张信用卡的大小，售价也与前代相同，仅为 35 美元。而体积更为小的树莓派 Zero 也配置了强劲的 1GHz 单核 CPU，RAM 达到了 512MB，售价更是惊人低的 5 美元。树莓派 Zero 体积小、性能强，这也使它可以被用于一些特殊的情形，例如用于可穿戴设备。

### 1.3.2. 易于教学

树莓派一直以来都很适合用于教学，它可以用来促进校内基础的计算机科学技能的培养。由于树莓派的成本低廉，这类微型计算机也将很有市场。树莓派已经在学校里成为了一个教授计算机科学的主要平台选择。这款设备在大学里很受学生的欢迎，它们被用于控制机器人或者被用于智能家居项目。南安普顿大学的教授 Simon Cox 还将 64 个树莓派链接起来制作成了一个可以处理复杂计算任务的超级计算机。

### 1.3.3. 易实现的功能性

目前已经有将近上千万个树莓派被售出，而在电子与计算机实验室中人们经常可以看到树莓派的身影。就在去年，树莓派打破了 Amstrad PCW 的记录，创造了全英国最畅销计算机的新纪录。

树莓派这样的微型计算机在价格上也在持续走低，这将有助于全世界人们对这类计算机的使用。即使是发展中国家的人们也可以使用计算机，也可以接入网络，从而享受数字世界带来的好处，世界经济也会因此更加繁荣。

如今树莓派的处理能力已经绝非仅仅是开关灯那么简单。然而虽然每一代树莓派都带来了新的能力，我们还是没能完全开发出树莓派的潜能，因此人们还需

要多想用途，然后利用树莓派来实现它们。

## 1.4. 总结

通过对树莓派和物联网的概念介绍和初步了解，作为新兴硬件和新兴网络概念，他们俩的结合肯定带来很大的应用前景和广阔市场份额。

艾研信息利用开发园区的地理位置优势，与国内多所院校建立合作技术关系，致力于为高校、科研院所、企业提供全方位解决方案。主要产品包括高校教育实验平台、面向高校研究所应用开发方案等。针对于越来越兴起的物联网概念，各大高校纷纷开设了物联网专业来满足社会对于专业物联网人才的需求，艾研也积极的响应这一号召，需要在物联网上一些积淀来促进高校对于物联网人才的培养。

综上所述，开发一款基于树莓派的物联网套件并对其做一定的研究和编写一些教材资料是具有重要的意义和很高的商业价值的！

# 2. 基础储备

基于树莓派上的物联网开发不同于类似 TI MCU MSP430LaunchPad 的开发，只需要有一台装有 CCS 的 PC 和 LauchPad 就可以做板载调试开发。树莓派上物联网开发需要更多的软硬件环境作为开发保障，并且对于开发者提出了更高的要求。

学习者不仅仅需要掌握基本的 C 语言基础知识，而且也需要知道类似于 python 编程，javacript 语言，网络协议（HTTP/CoAP/REST 等等）众多的跨学科跨领域知识储备。对于那些认为 C 语言、寄存器操作、16 进制协议可以解决一切问题的嵌入式工程师来说，python 和 javacript，REST 和 CoAP 协议显得臃肿不堪并毫无用处。但是事实并非如此。在接下来的讲解中我们就能慢慢体会它们的强大功能。

作为基础准备当然包括硬件物质和软件知识体系上的准备，我们将其一分为二分别加以介绍和准备：

## 2.1. 硬件准备

### 2.1.1. 树莓派（Raspberry 3）以及附属配件设备

#### 2.1.1.1. 树莓派

只是一块开发板，如下图所示：

图 2-1 树莓派主板

四核 1.2GHz Broadcom BCM2837 64 位 CPU 1GB RAM 板载 BCM43143 WiFi 和蓝牙低能耗(BLE) 40 引脚扩展 GPIO 4 个 USB 2 端口 4 路立体声输出和复合视频端口 全尺寸 HDMI CSI 照相机端口用于连接树莓派照相机 DSI 显示端口用于连接树

莓派触屏显示器。推荐开发者搜索淘宝“开源硬件商城”，通过权威代理渠道购买正品开发板学习和开发。（https://ivision-openhw.taobao.com/）

#### 2.1.1.2. 电源适配器


图 2-2 电源适配器

树莓派 3 代官方推荐电源 5V2.5A，如果负载不大的情况也可以选择 5V2A 电源，作为树莓派主板运行的发动机，电源的选择尤为重要。

https://item.taobao.com/item.htm?spm=2013.1.0.0.B5j1Ub&id=527761239507

#### 2.1.1.3. TF 闪卡

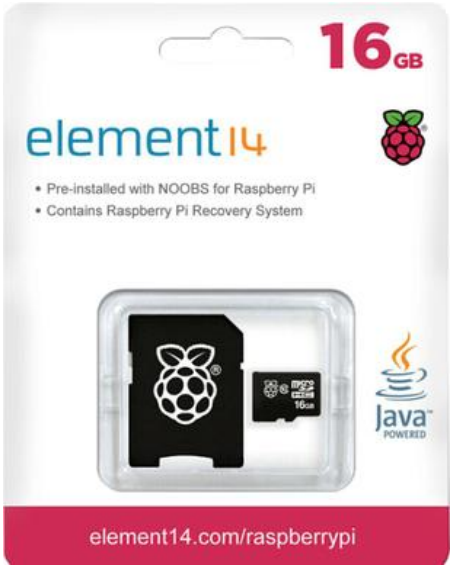



图 2-3 SD 闪卡

Raspbain 为用于运行各种树莓派系统的最快方式. 此 16GB 的微型 SD 卡可驱动多个操作系统, 用于可快速便捷地启动树莓派.可以通过运行 sudo apt-get 更新和 sudo apt-get 升级确保您的操作系统为最新版.

https://item.taobao.com/item.htm?spm=2013.1.0.0.B5j1Ub&id=536538755379

#### 2.1.1.4. 散热片

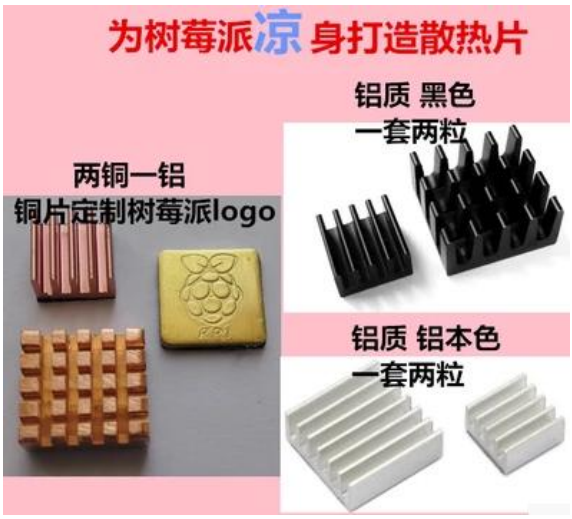



图 2-4 散热片

众所周知由于树莓派 3 主芯片主频提高到了 1.2GHz，性能大幅度提高的同时，如何给树莓派散热成为困扰广大树莓派 3 使用者的一个主要问题，为了解除大家的后顾之忧，树莓派 3 需要一组散热片套件为其降温。

https://item.taobao.com/item.htm?spm=2013.1.0.0.B5j1Ub&id=44427919831

#### 2.1.1.5. 触摸屏

图 2-5 触摸屏显

树莓派 7 寸触摸屏帮助客户有能力制造 all-in-one 一体机，完成类似平板电脑、信息娱乐系统和嵌入式设计等项目。触摸屏通过一个提供电源和信号的转接板到树莓派主板，像素为 800*600，它就是一个交互式的设备，最新触摸屏软件驱动程序将支持虚拟屏幕上的键盘，不需要额外插入一个键盘和鼠标。

https://item.taobao.com/item.htm?spm=2013.1.0.0.B5j1Ub&id=522575645997

## 2.2. 物联网实验套件 AY-IOT KIT

### 2.2.1. 套件概述

作用物联网应用的创新设计，如果在众多环节中出现一个环节的问题，都将导致最终创新想法不能实现。本实验套件设法的目标就是提供物联网创新应用设计的学生，学习物联网开发的各种工具和了解其基本原理，掌握各个环节的快速设计工具的综合应用方式，为快速实现自己的创新想法而做好准备。

物联网实验套件同时兼容 TI CC3200Launchpad 和 Raspberry 3，方便不同应用背景的用户学习使用；实验套件提供完整的全套代码和应用视频指导，回避繁琐的指导书（当然也提供指导书方便不同偏好的学习开发者）；套件的模块设计了标准的接口，方便套件的扩展，同时也便于用户自行扩展；因为手机品种过多，实验套件手机端不提供专门 APP 的程序，但提供了手机网页版的程序，用户可采用套壳的方式快速实现可以个人手机上呈现的 APP。


图 2-6 大学生创新实践套件

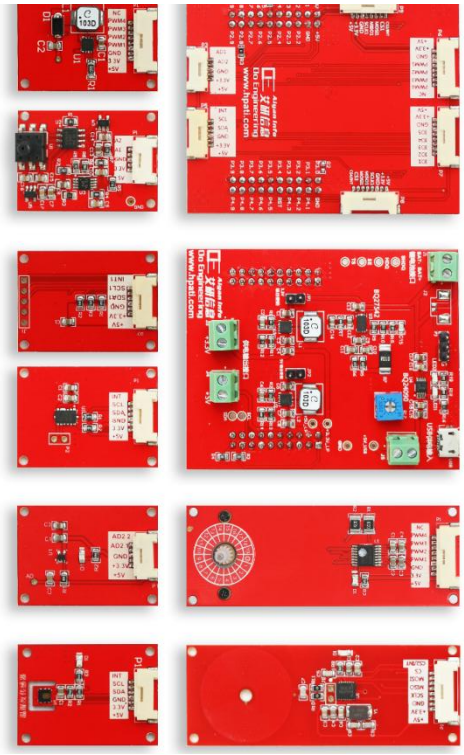


图 2-7 套件展示

而想要链接到 Raspberry3 上，除了上述的硬件配饰外，艾研还针对 Raspberry3 的通用接口设计了一块通用接口转接板，


图 2-8 Raspberry 3 转接板

它含有如下一些通用接口：

- 1 个 GPIO 接口，包含 5 个 GPIO 口
- 1 个 PWM 接口，包含 4 个 PWM 端口
- 1 个 SPI 通信接口

- 1 个 AD 接口，包含 1 个 16 位采样通道（ADS1100）
- 1 个 I2C 通信接口

原理图如下：

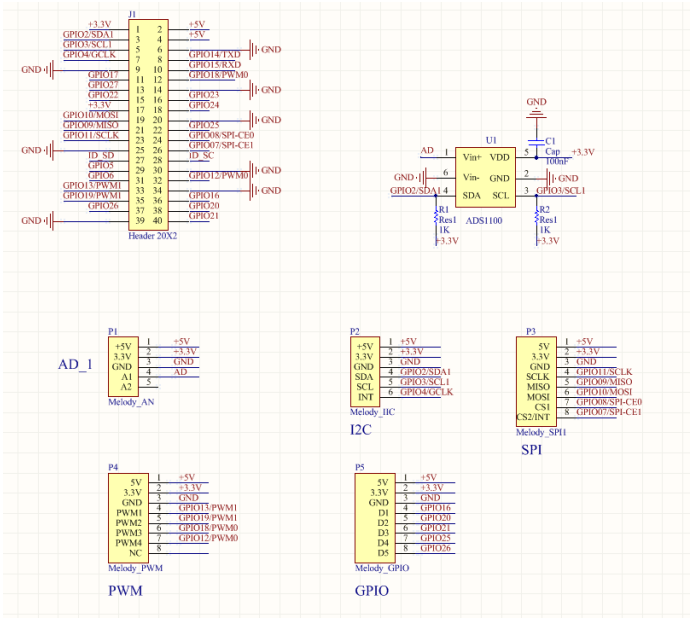



图 2-9 Raspberry 3 转接板原理图

实现的原理也很简单，就是把 Raspberry 3 的通用 40 针管脚根据物联网模块的需要单独的链接出来方便使用，其中增加了一块 ADS1100 芯片的目的主要是为了 LMT84AD 转换模块的使用。考虑到 Raspberry 不带有模拟输入接口，所有的接口都是数字接口，所以模拟量只能通过这个转接板上的 ADS1100 AD 功能转化为数字量以 I2C 通信的形式传输给 Raspberry 主板。

## 2.2.2. 实验列表

整个套件涵盖了物联网概念的大部分元素，有关于 IO 控制、I2C/SPI 传感器通信、PWM 电机控制等环节。

表 2-1 实验套件模块列表

| 模块名称 | 器件 | 例程 | 功能 |
|---|---|---|---|
| 三轴加速度模块 | ADXL345 | ADXL345 | 使用 I2C 通信完成 ADXL345 寄存器数据的读写读取 XYZ 三轴数据并显示在网页端 |
| 温湿度传感器模块 | HDC1080 | HDC1080 | 使用 I2C 通信完成 |

| | | | |
|---|---|---|---|
| | | | HDC1080 寄存器数据的读写读取温湿度数据并显示在网页端 |
| 模拟温度模块 | LMT84 | LMT84 | 读取并转换温度数据，显示在网页端 |
| 光感模块 | OPT3001 | OPT3001 | 使用 I2C 通信完成 OPT3001 寄存器数据的读写读取并转换光强数据，显示在网页端 |
| 高亮 LED 驱动模块 | TPS61043LED | TPS61043LED | 网页显示 LED 调节滚轮，使用滚轮调节 PWM 占空比，改变 LED 亮度 |
| 步进电机模块 | DRV8833 步进电机 | DRV8833 步进电机 | 使用 PWM 控制步进电机转速和方向 |
| LDC1000 模块 | LDC1000 | LDC1000 | 通过 SPI 完成 LDC1000 寄存器读写读取接近数据 Proximity 和频率 Frequency Counter 计算电感 |
| 血压模块 | MPS-3117 | BloodPressureSample | 使用 AD 完成静态压和血压波动数据采集完成与上位机程序通信 |
| 电池管理模块 | BQ24090<br>BQ27542 | LiBattery | 使用 I2C 完成 BQ27542 寄存器读写监控锂电池状态 |

## 2.3. 开发用 PC

除了树莓派和实验套件之外当然还需要一台主机作为开发用的平台来搭建物联网开发的基本开发环境。

## 2.4. 软件安装

硬件介绍完成后，在我们的硬件平台下还需要搭建响应的软件开发环境。由于硬件分为两块内容，即基于 PC 的软件和基于树莓派上的软件。我们一一作如下介绍：

### 2.4.1. PC 端软件安装

PC 端软件主要考虑从拿到一块树莓派裸板为起始点，要通过如下一些软件的操作才能完成系统正常运行起来。

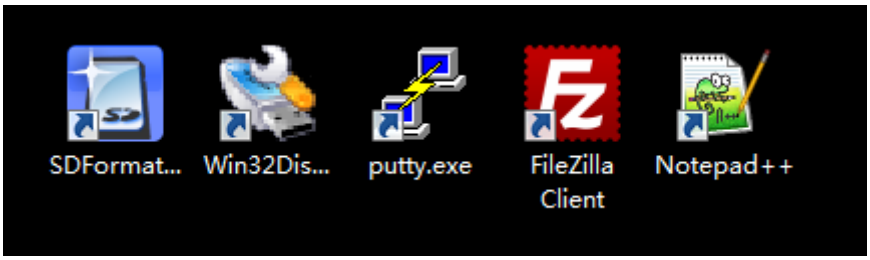



图 2-10 PC 软件需求

#### 2.4.1.1. SD 卡格式化-(SDFormatter)

树莓派的系统由上面的介绍可知它是运行在一张 TF SD 卡上面，假如用户手上有张空白的 SD 卡，首选第一步需要做的就是第一步格式化 SD 卡。如下图所示，只需选择你需要格式化的驱动盘点击格式化即可！


图 2-11 SDFormatter 软件界面

### 2.4.1.2. 树莓派系统烧写-(Win32DiskImager)

树莓派操作系统可以通过访问树莓派官方网站(https://www.raspberrypi.org/downloads/raspbian/)下载。当然如果发现访问不了可能还需要用到 Lantern 之类的翻墙软件。自行下载后安装运行后翻墙后再访问即可。最新版本已经到了 2017-03-02-raspbian-jessie.zip

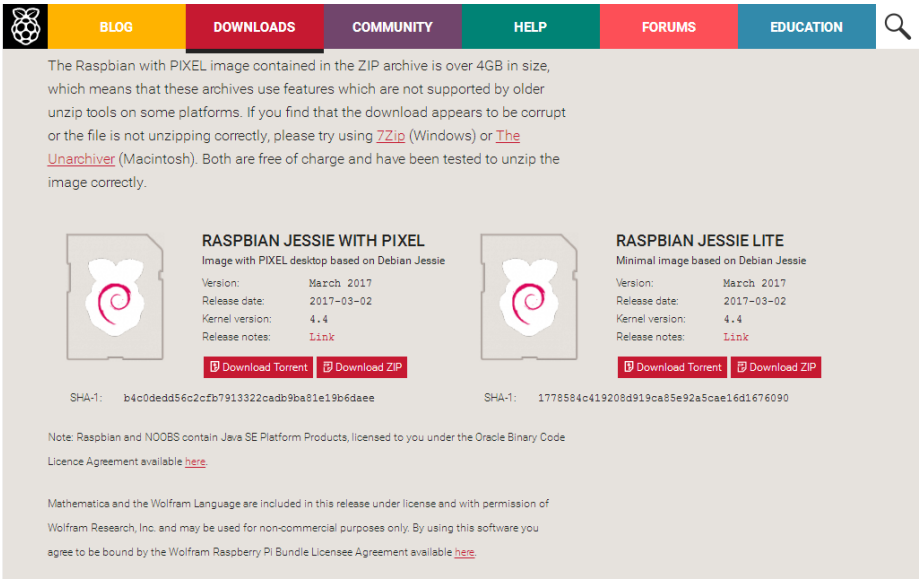



图 2-12 Raspbian 操作系统下载

下载完成后，得到的是一个 zip 压缩包在当前路径下解压会生成一个.img 文件这就树莓派系统的镜像文件。打开 Win32DiskImager 软件，打开文件夹选中解压后的 img 文件，并选中 SD 卡所在的盘符如下图所示，点击 Wirte 就是写 img 文件到 SD 中去，可以理解为将树莓派系统安装在 SD 上。

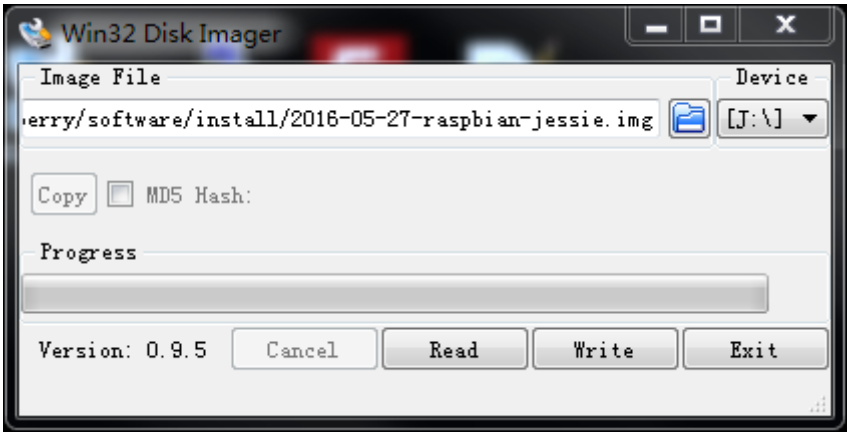



图 2-13 Raspbian 操作系统安装到 SD

### 2.4.1.3. 远程访问-(Putty)

完成 SD 卡的树莓派系统烧写后，将完成的带树莓派操作系统的 SD 卡插入树莓派卡槽中，链接供电将树莓派上电。如果连接了触屏或者其他显示器的情况下，能够看到树莓派系统启动界面。

图 2-14 Raspbian 操作系统启动界面显示

如果没有链接任何显示设备也可以通过 Putty 这种远程访问的方式进入树莓派命令行显示模式。当然前提是要知道树莓派的 IP 地址，我们可以通过 LanView 之类的局域网 IP 扫描方式查看树莓派的 IP 地址。


图 2-15 LANView 扫描局域网查看树莓派端口

如果连接了触屏由于 Raspberry 3 自带 WiFi 模块的，也可以配置无线网络连接到局与无线网络中。例如树莓派的 IP 地址是 192.168.1.117，打开 Putty 软件

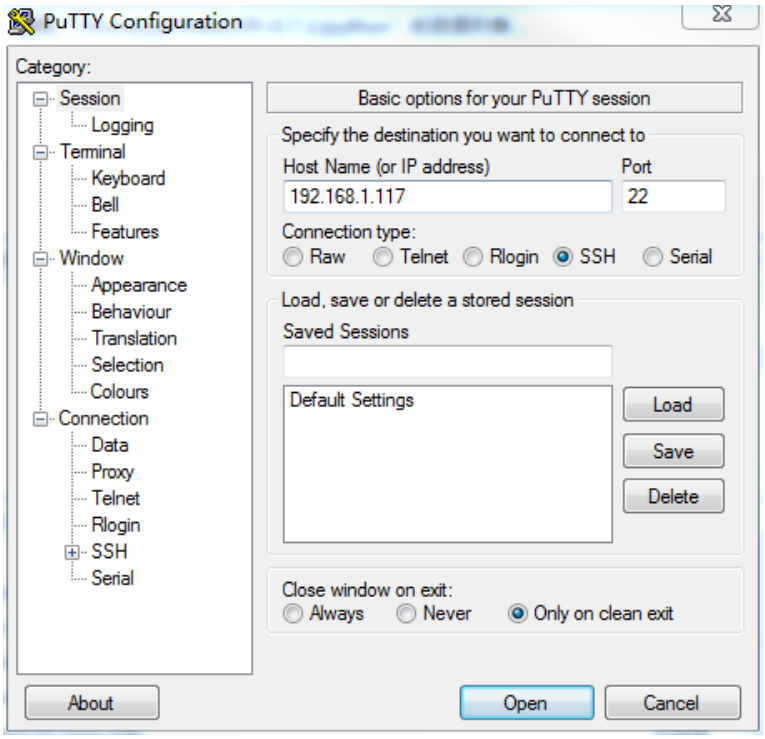



图 2-16 Putty 访问树莓派

- Host Name 输入：192.168.1.117

- Port 保持 ： 22
- Connection type： 保持 SSH

点击 Open 按钮

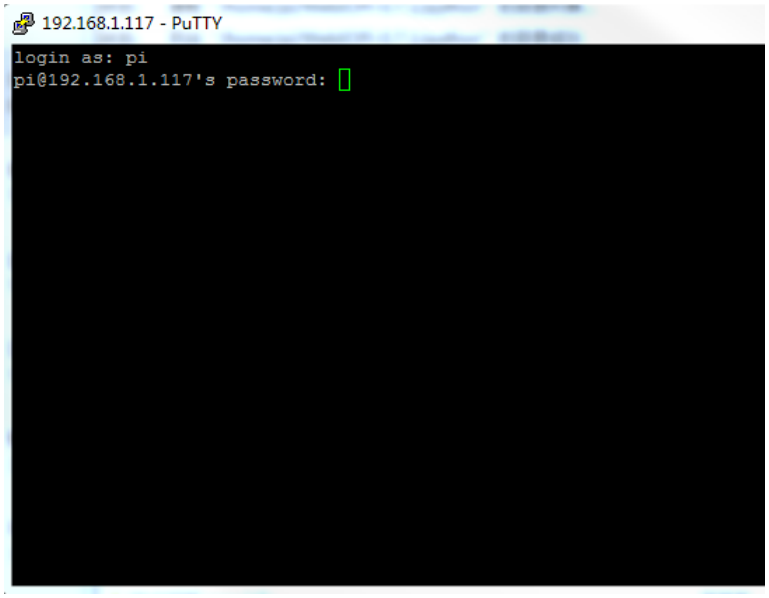



图 2-17 Putty 访问树莓派登陆界面

需要输入访问有户名和密码，初装的树莓派系统都可以通过用户名：pi 密码：raspberry 来访问登陆。需要注意的是，在输入密码的时候它并不会显示任何字符，只要输入完成和正确点击回车便能进入系统了。

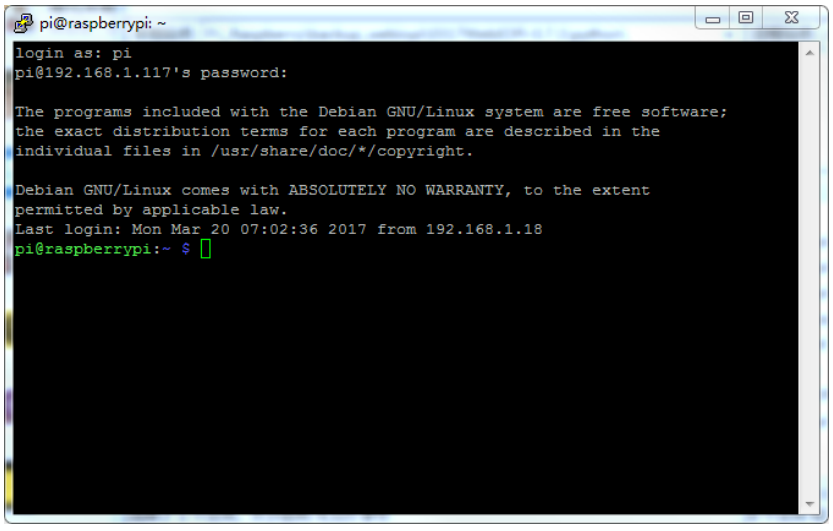



图 2-18 Putty 登陆 Raspberry

### 2.4.1.4. 文件共享-(FileZilla)

FileZilla 是一个免费开源的 FTP 软件，分为客户端版本和服务器版本，具备所有的 FTP 软件功能。可控性、有条理的界面和管理多站点的简化方式使得

Filezilla 客户端版成为一个方便高效的 FTP 客户端工具，而 FileZilla Server 则是一个小巧并且可靠的支持 FTP&SFTP 的 FTP 服务器软件。我们使用客户端版本即可。

主要目的在于通常的开发诸如代码编写或者脚本编写还是在 PC 端进行，毕竟树莓派上是 linux 系统不管 UI 操作体验还是编辑软件功能还是不如 Windows 操作系统那么方便。但是运行和测试需要下载到树莓派一端。通过 Filezilla 我们能够上传和下载树莓派上的数据和程序等。

软件的下载和安装就不在赘述了，自行百度即可。

安装完成后打开软件：（注：在使用本软件之前首先要确认树莓派系统上已经安装运行了 tftp 服务,tftp 安装会在后续 Raspberry 软件里面介绍）

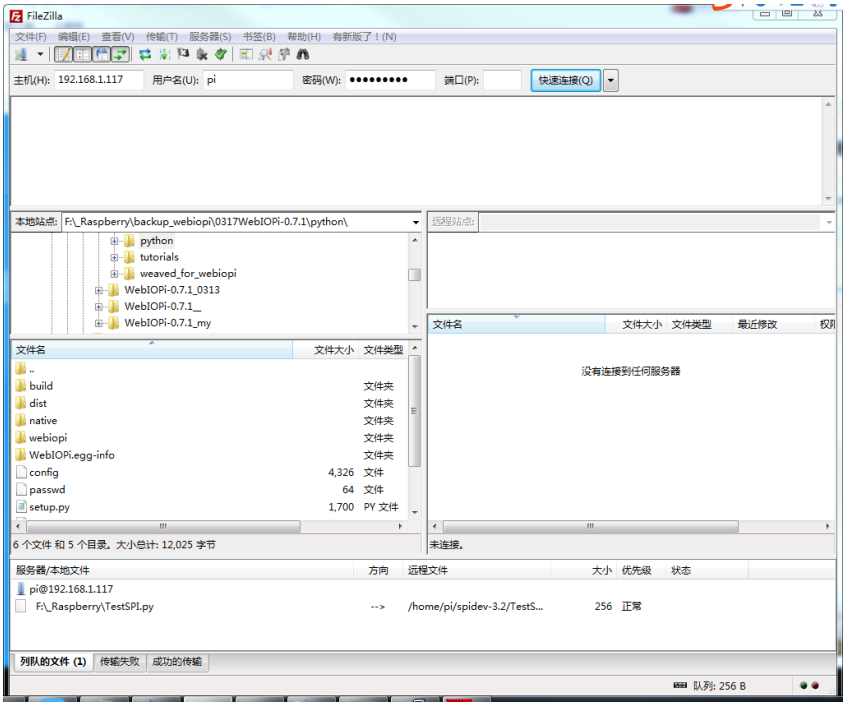



图 2-19 filezilla 登陆 Raspberry

如上图所示，

- 主机栏输入树莓派 IP 地址：192.168.1.117
- 用户名：pi
- 密码：raspberry
- 端口：空置

点击快速连接即可！

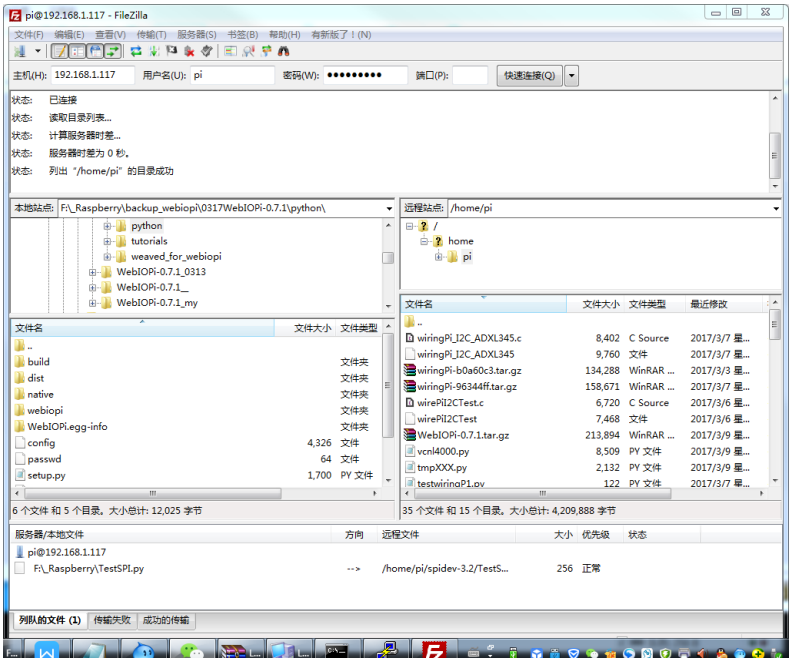



图 2-20 filezilla 上的 Raspberry 文件系统

在状态栏中会显示有没有正常的链接到树莓派，目前显示登陆到来树莓派的/home/pi 目录，即用户 pi 的用户目录。在右侧的文件夹系统显示的即是板载的文件系统，左侧显示的是 PC 端的文件系统。

### 2.4.1.5. 程序编辑-(NotePad++/SourceInsight)

强大无比的 Notepad++，指代码编辑器或 WINDOWS 中的“记事本”程序。在 WINDOWS 下主要用于文本编辑。它是一款开源、小巧、免费的纯文本编辑器。在文字编辑方面与 Windows 写字板功能相当。当然，更重要的是 Notepad++更是程序员们编写代码的利器！（小编特别喜欢使用它）关于下载和安装自行百度即可。

本教程中主要涉及了很多 Python、纯 C 语言或者脚本等存文本代码和数据，使用 NotePad++很方便。如下图所示：

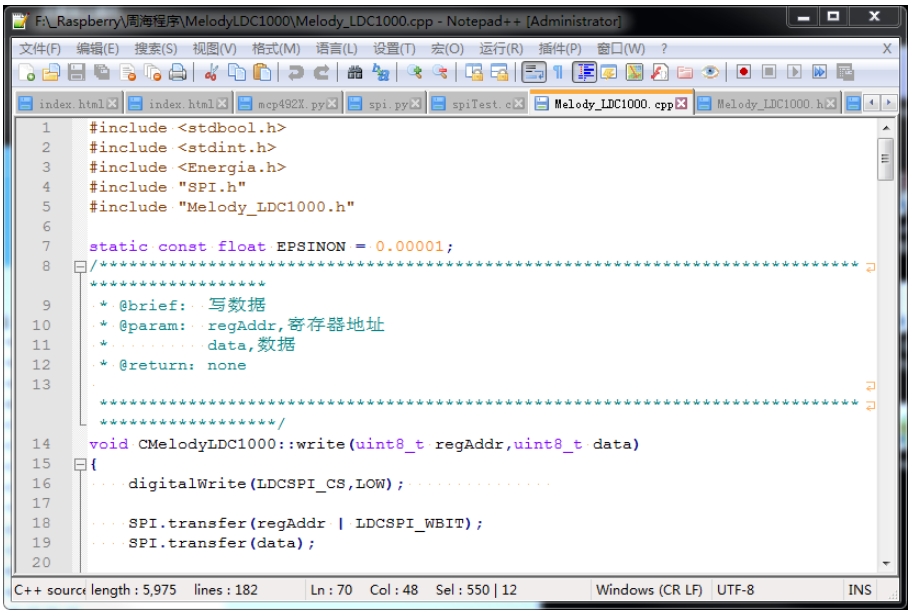



图 2-21 NotePad++使用

## 2.4.2. 树莓派端软件安装

树莓派操作系统 Raspbian 当然自带了很多强大的软件，但是并不是本文着重介绍的，基于物联网的开发需要在树莓派上安装很多相应的软件配置。本章节将会逐一进行介绍，首先当然是 Raspbian 操作系统本身。

### 2.4.2.1. 树莓派操作系统-Raspbian-jessie

Raspbian(单纯的 Arm 版的 Linux 系统，基于 Debian)是当前实用最广泛的操作系统。作为树莓派的操作系统，它是运行系统的基础。但是在本章节没有需要特别关注的内容。只需要能够安装 PC 端软件配置内容将操作系统正确烧写到 SD 卡中正常运行即可。

完成系统安装后，运行$sudo apt-get update 更新系统。

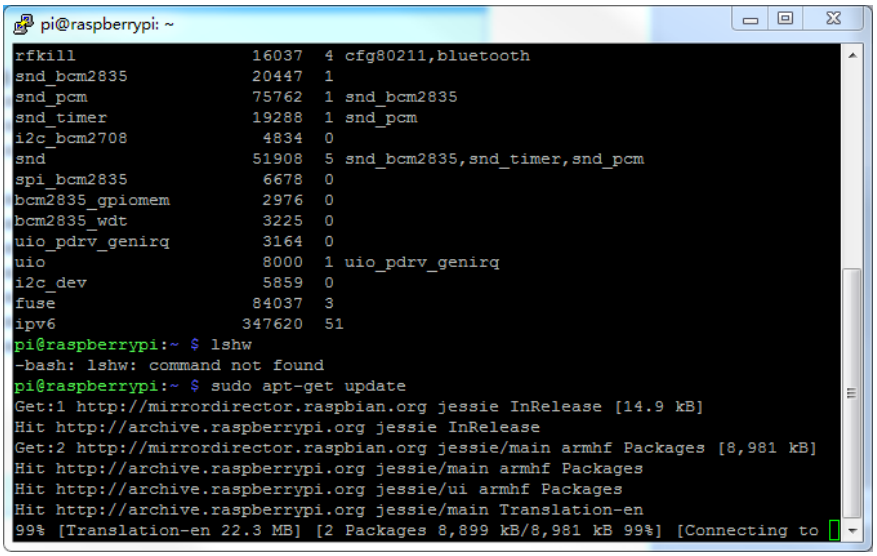



图 2-22 更新操作系统

### 2.4.2.2. FTP 服务-(vsftpd)

上文中我们介绍过了 PC 端的 filezilla 软件的安装其中提到需要在 Raspberry 端安装 FTP 服务器。如果想在 Raspberry 上搭建一个安全、高性能、稳定性好的 FTP 服务器，那么 vsftpd 可能是首选应用。vsftpd 意思为“very secure FTP daemon(非常安全的 FTP 进程)”，是一个基于 GPL 发布的类 UNIX 类操作系统上运行的服务器的名字（是一种守护进程），可以运行在诸如 Linux、BSD、Solaris、HP-UX 以及 Irix 等系统上面。vsftpd 支持很多其他传统的 FTP 服务器不支持的良好特性。

安装步骤如下：

- **安装 vsftpd 服务器**

在树莓派命令行环境下运行如下命令安装 vsftpd 服务器：

```
pi@raspberrypi:~ $ sudo apt-get install vsftpd
```

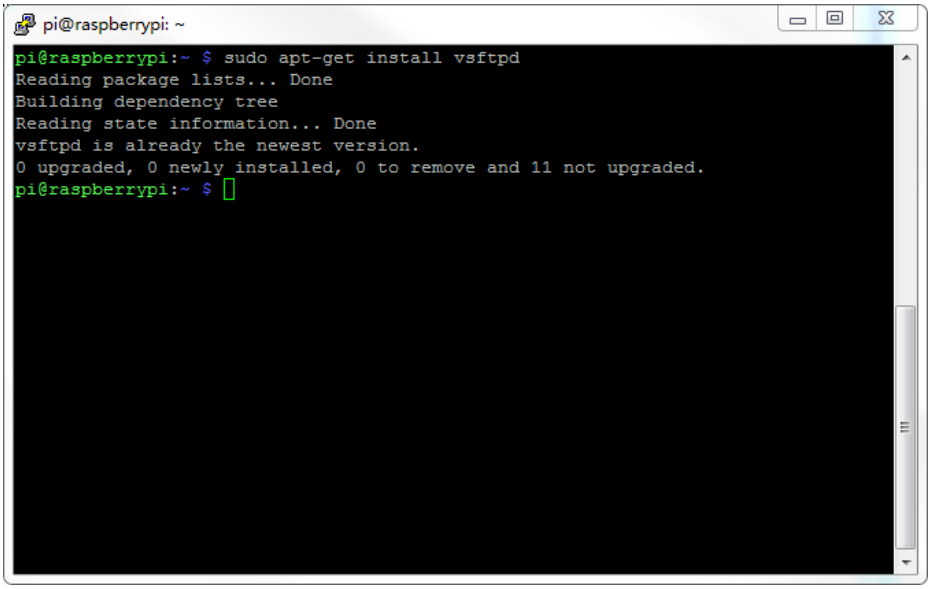



图 2-23 安装 vsftpd 服务器

- **启动 ftp 服务**

pi@raspberrypi:~ $ sudo service vsftpd start

- **编辑修改 vsftpd 的配置文件**

pi@raspberrypi:~ $ sudo nano /etc/vsftpd.conf

使用 nano 工具打开 vsftpd.conf 文件并修改其中的如下内容（这里能够体会到 linux 下编辑的痛苦！如果不了解 nano 的使用可以自行百度了解）：

找到以下行，定义一下 anonymous_enable=NO

表示：不允许匿名访问 local_enable=YES 设定本地用户可以访问。

write_enable=YES 设定可以进行写操作。如图所示：

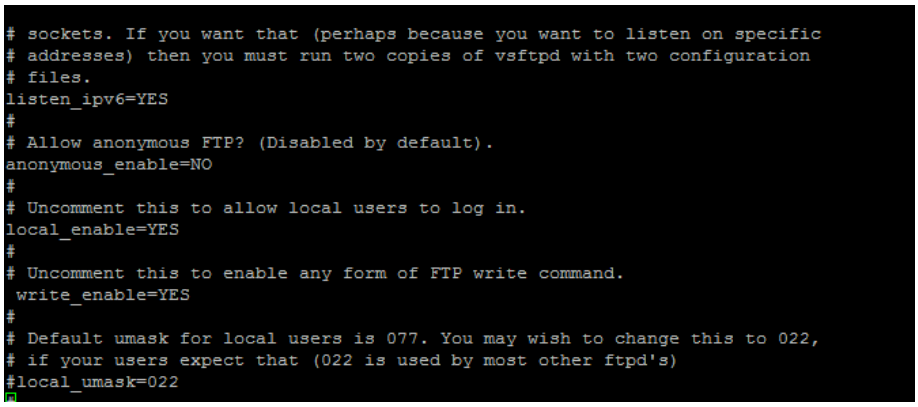



图 2-24 修改 vsftpd.conf 内容

- **重新启动 vsftpd 服务：**

pi@raspberrypi:~ $ sudo service vsftpd restart

➢ **测试连接状态，**

通过 2.4.1.4 FileZilla 章节介绍测试树莓派通信的联通状态。

### 2.4.2.3. 开发语言-Python

Python 是一种面向对象的解释型计算机程序设计语言。Python 是纯粹的自由软件， 源代码和解释器 CPython 遵循 GPL(GNU General Public License)协议。Python 语法简洁清晰，Python 具有丰富和强大的库。它常被昵称为胶水语言，能够把用其他语言制作的各种模块（尤其是 C/C++）很轻松地联结在一起。常见的一种应用情形是，使用 Python 快速生成程序的原型（有时甚至是程序的最终界面），然后对其中有特别要求的部分，用更合适的语言改写，而后封装为 Python 可以调用的扩展类库。需要注意的是在您使用扩展类库时可能需要考虑平台问题，某些可能不提供跨平台的实现。

```
pi@raspberrypi:~ $  sudo apt-get install python-smbus
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-smbus is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 11 not upgraded.
pi@raspberrypi:~ $
```

图 2-25 Raspberry 安装 python

安装完成可以测试是否有安装成功，直接输入 Python 就进入 python 界面变成”>>>”，显示目前安装的是 python2.7.9 的版本号。Ctrl+Z 能够退出 Python，回到树莓派命令行输入。

```
pi@raspberrypi:~ $ python
Python 2.7.9 (default, Sep 17 2016, 20:26:04)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

图 2-26 Raspberry 运行 python

### 2.4.2.4. I2C 检测工具-(i2c-tool)

在进行 I2C 相关程序开发时，很多时候我们需要确认硬件是否正常连接，设备是否正常工作，设备的地址是多少等等，这里就需要使用一个用于测试 I2C 总线的工具——i2c tools，下面我们就对这个工具的安装和使用做个简单介绍。

## 2.4.2.4.1. 使能树莓派的 I2C 功能

安装 I2C-Tool 之前请确认设备的 I2C 功能已经开启，在命令行输入

```
pi@raspberrypi:~ $ sudo raspi-config
```

可以打开树莓派软件配置工具

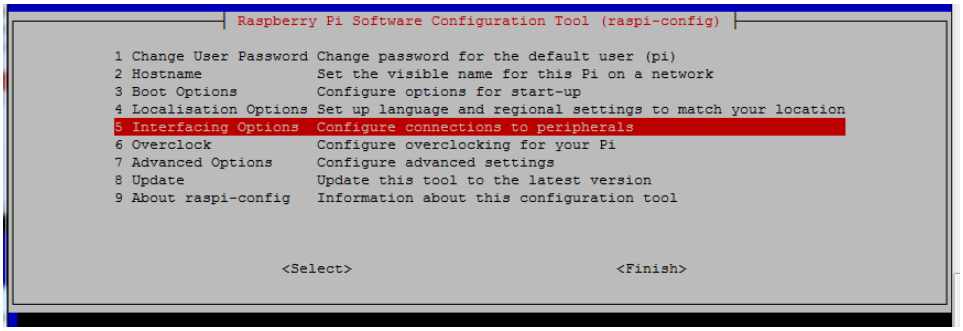



图 2-27 Raspberry 软件配饰工具

选择 Interfacing Options（各种接口配置）

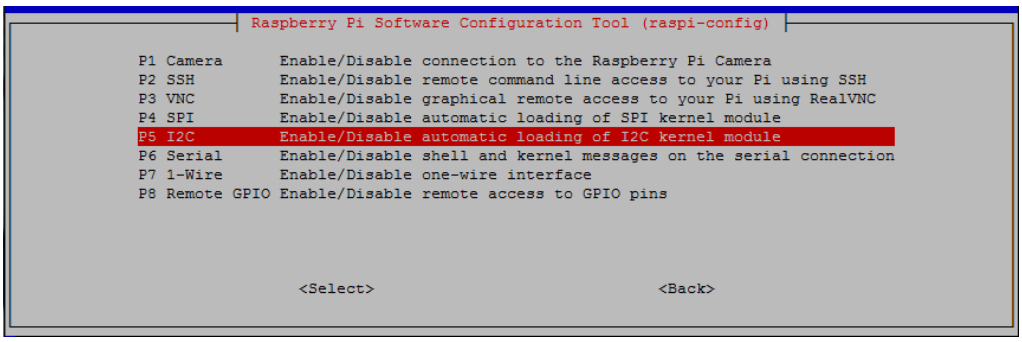



图 2-28 I2C 功能选项

选择 I2C 配置选项并且选择<YES>使能 I2C 功能后退出回到命令行输入界面

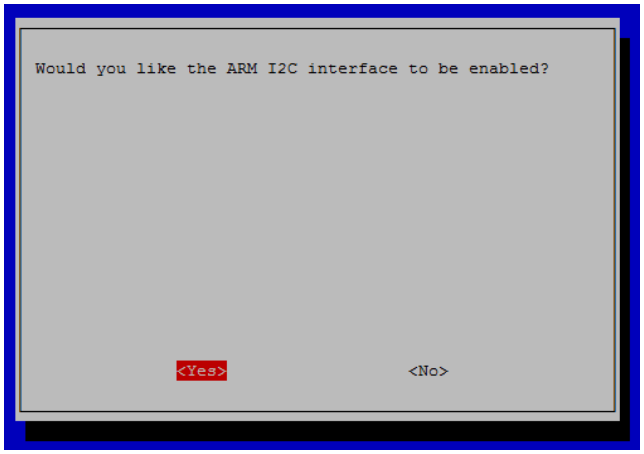



图 2-29 I2C 使能

然后输入 sudo reboot 重新启动系统是最新的配置生效。

```
pi@raspberrypi:~ $ sudo reboot
```

### 2.4.2.4.2. 安装 I2C-tool

在控制台输入$sudo apt-get install i2c-tools （前提条件必须是树莓派已经成功连接网络了）或者也可以使用线下安装方式，自行下载本地版程序通过 ftp 工具上传到树莓派后解压安装。

```
pi@raspberrypi:~ $ sudo apt-get install i2c-tools
```

命令即可安装 i2c-tools，安装完成后可以使用$sudo i2cdetect -l 查看安装状态

```
pi@raspberrypi:~ $ sudo i2cdetect -l
i2c-1   i2c             3f804000.i2c                            I2C adapter
pi@raspberrypi:~ $
```

图 2-30 I2C 查看

如果看到如上的显示内容表示安装成功。

### 2.4.2.4.3. I2C-tool 使用

- I2C 设备查询：sudo i2cdetect -y 1 命令即可扫描接在总线上的所有 I2C 设备，并打印出该设备的 I2C 总线地址。

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

图 2-31 Raspberry I2C 总线扫描

使用该命令输入了两个参数，第一个参数为可选参数，若不传入这个参数，则执行该命令的时候会有一些提示，这里可以带上该参数，第二个参数是 I2C 设备号，由于我们系统中只有一路 I2C 总线，且设备号为 1，所以我们这里传入 1。执行该命令后会输出总线上 I2C 设备的总线地址信息，比如我接了一个 AD 传感器，则扫描到的设备地址为 0x48，还有一个温湿度传感器设备地址是 0x40。这和传感器手册中所给的地址匹配。

- 寄存器数据导出：

sudo i2cdump -y 1 0x48 即可导出 I2C 设备寄存器中的数据

-y　　代表取消用户交互过程，直接执行命令

1　　代表 I2C 设备号

0x48　代表 I2C 设备地址

该命令会到处 0x00 到 0xFF 地址范围内的所有数据，输出结果如下图所示

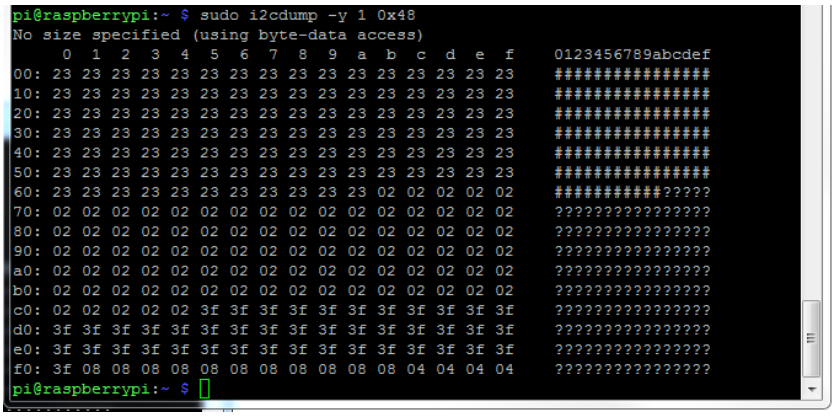



图 2-32　I2CDump 使用

- 寄存器数据写入：

sudo i2cset -y 1 0x48 0x90 0x55 即可向设备地址为 0x48 的 0x90 寄存器地址写入 0x55

-y　　代表取消用户交互过程，直接执行命令

1　　代表 I2C 设备号

0x48　代表 I2C 设备地址

0x90　代表寄存器地址

0x55　代表向寄存器写入的数据

- 寄存器数据读出

sudo i2cget -y 1 0x48 0x90

即可读出总线地址为 0x48 设备的 0x90 寄存器地址的数据

-y　　代表取消用户交互过程，直接执行命令

1　　代表 I2C 设备号

0x48　代表 I2C 设备地址

0x90　代表寄存器地址

## 2.5. 总结

本章节主要为学习树莓派准基础储备工作，主要面向对象是高校的学生或者对于树莓派和物联网本身没有太多基础的初学者而设定，假如您对这些内容已经有了一定的了解或者熟悉的话，可以自行跳过本章节的介绍。

在这章节中，主要是对树莓派和物联网这两个概念做了描述，并且从学习的角度分析如果要了解它们，需要开发者不管从硬件和软件方便做好如上的准备才

能继续后续的学习开发工作。

# 3. wiring PI

## 3.1. 简介

wiringPi 是应用于树莓派平台的 GPIO 控制库函数，wiringPi 遵守 GUN Lv3。wiringPi 使用 C 或者 C++开发并且可以被其他语言包转，例如 python、ruby 或者 PHP 等。wiringPi 中的函数类似于 Arduino 的 wiring 系统，这使得熟悉 arduino 的用户使用 wringPi 更为方便。

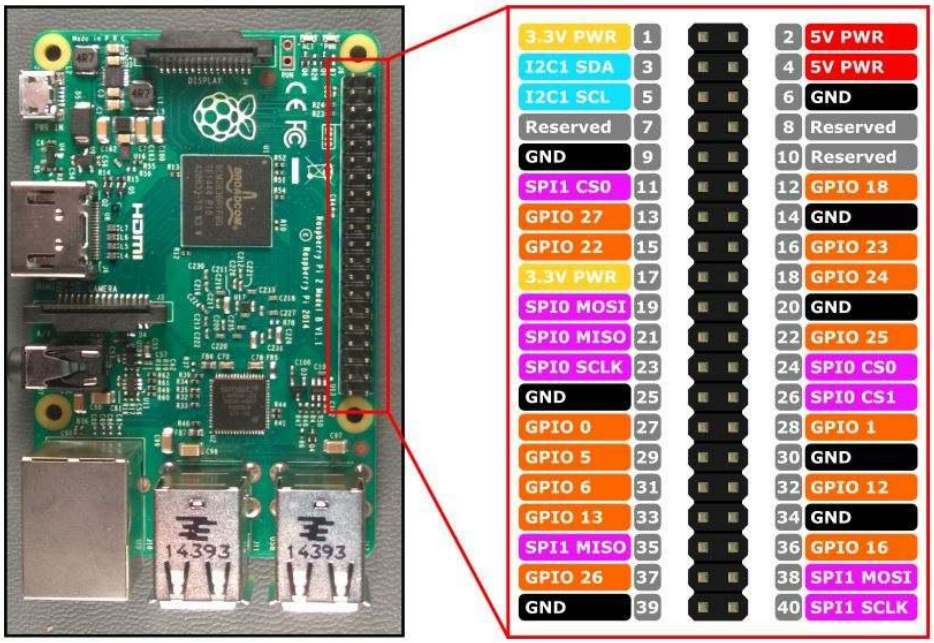



图 3-1 Raspberry 接口扩展

树莓派具有 26 个普通输入和输出引脚（GPIO）。在这 26 个引脚中具有 8 个普通输入和输出管脚，这 8 个引脚既可以作为输入管脚也可以作为输出管脚。除此之外，树莓派还有一个 2 线形式的 I2C、一个 4 线形式的 SPI 和一个 UART 接口。树莓派上的 I2C 和 SPI 接口也可以作为普通端口使用。如果串口控制台被关闭便可以使用树莓派上的 UART 功能。如果不使用 I2C，SPI 和 UART 等复用接口，那么树莓派总共具有 8+2+5+2 =17 个普通 IO。wiringPi 包括一套 gpio 控制命令，使用 gpio 命令可以控制树莓派 GPIO 管脚。用户可以利用 gpio 命令通过 shell 脚本控制或查询 GPIO 管脚。wiringPi 是可以扩展的，可以利用 wiringPi 的内部模块扩展模拟量输入芯片，可以使用 MCP23x17/MCP23x08（I2C 或者 SPI）扩展 GPIO 接口。另外可通过树莓派上的串口和 Atmega（例如 arduino 等）扩展更多的 GPIO 功能。另外，用户可以自己编写扩展模块并把自定义的扩展模块集成到 wiringPi

中。wiringPi 支持模拟量的读取和设置功能，不过在树莓派上并没有模拟量设备。但是使用 wiringPi 中的软件模块却可以轻松地应用 AD 或 DA 芯片。

wiringPi 适合那些具有 C 语言基础，在接触树莓派之前已经接触过单片机或者嵌入式开发的人群。wiringPi 的 API 函数和 arduino 非常相似，这也使得它广受欢迎。作者给出了大量的说明和示例代码，这些示例代码也包括 UART 设备，I2C 设备和 SPI 设备等，毫无疑问地说 wiringPi 功能非常强大。

所以在本文将 wiringPi 作为一个过渡章节，给哪些熟悉 C 语言编程和嵌入式开发但是对于 Python 不够熟悉的开发者一个过渡，了解 Raspberry 的基本端口和功能。也为后一章节介绍 WebIOPi 打下一定的基础。

## 3.2. 安装 wiringPi

wiringPi 的安装存在方案 1 和方案 2。wiringPi 使用 GIT 工具维护个更新代码，但是如果处于一些其他原因不能使用 GIT，那么也可以使用方案 B 下载和安装 wiringPi。

### 3.2.1. 方案 A--使用 GIT 工具

如果在你的平台上还没有安装 GIT 工具，可以输入以下命令：

pi@raspberrypi:~ $$sudo apt-get install git-core

如果在这个过程中出现错误，尝试更新软件，例如输入以下指令：

pi@raspberrypi:~ $$sudo apt-get update

pi@raspberrypi:~ $$sudo apt-get upgrade

紧接着可以通过 GIT 获得 wiringPi 的源代码

pi@raspberrypi:~ $$git clone git://git.drogon.net/wiringPi

若需要更新 wiringPi。

pi@raspberrypi:~ $$cd wiringPi

pi@raspberrypi:~ $$git pull origin

进入 wiringPi 目录并安装 wiringPi

pi@raspberrypi:~ $$cd wiringPi

pi@raspberrypi:~ $$./build

build 脚本会帮助你编译和安装 wiringPi。

### 3.2.2. 方案 2——直接下载和解压

简单的输入以下网址：

https://git.drogon.net/?p=wiringPi;a=summary

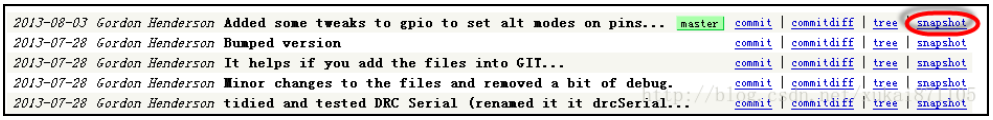



图 3-2 下载 wiringPi 最新版本源代码

点击图中的 snapshot 便可下载最新版本。可能下载到一个名为 wiringPi-98bcb20.tar.gz 的压缩包，需要助于的是 98bcb20 只是一个版本号，当你下载 wiringPi 的时候该版本号可能会发生改变。需要解压 wiringPi 源代码并进行安装，可输入以下指令：

pi@raspberrypi:~ $sudo tar xfz wiringPi-98bcb20.tar.gz

pi@raspberrypi:~ $cd wiringPi-98bcb20

pi@raspberrypi:~ $./build

需要注意的是，wiringPi 压缩包的名称很可能不是 98bcb20，请根据实际情况改变。它仅仅表示的是一个版本号。

## 3.3. 使用 wiringPi

### 3.3.1. 安装测试

测试 wringPi 是否安装成功。wiringPi 包括一套 gpio 命令，使用 gpio 命令可以控制树莓派上的各种接口，通过以下指令可以测试 wiringPi 是否安装成功。

输入：gpio -v

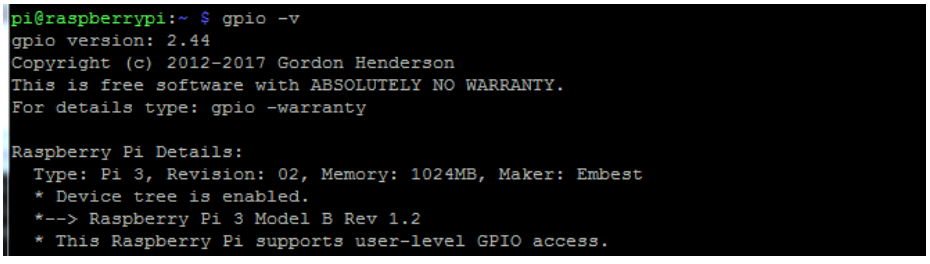



图 3-3 wiringPi 安装成功

输入：gpio readall

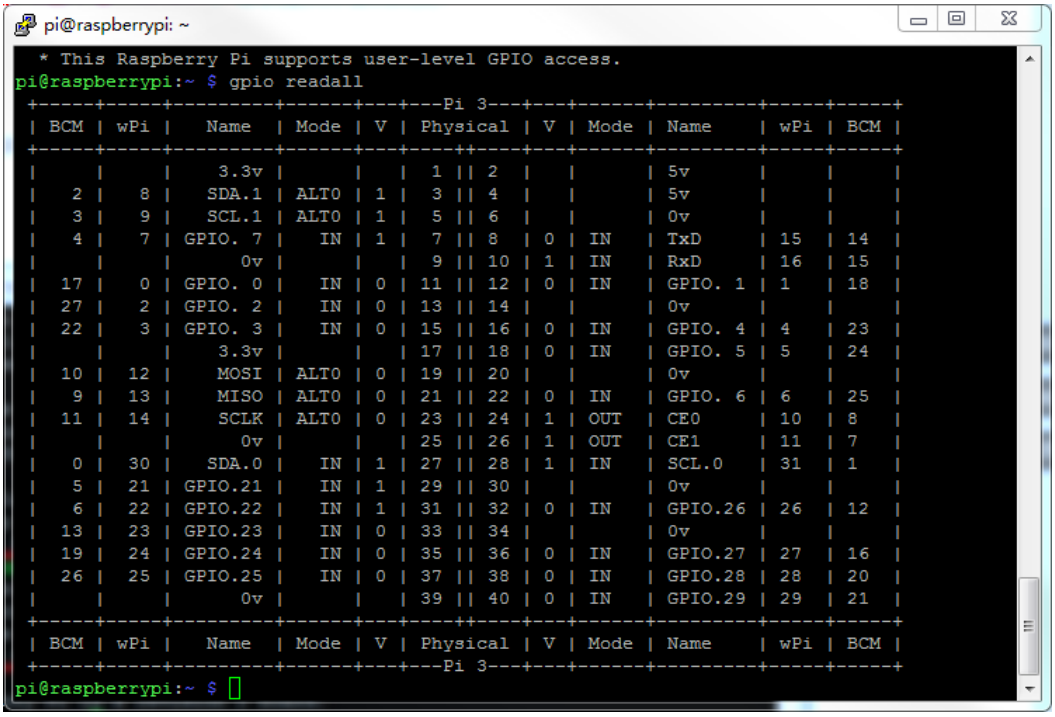



图 3-4 wiringPi GPIO 图谱

## 3.3.2. 引脚说明

上图是 wiringPi 的引脚说明，这里需要说明两点。第一树莓派存在版本 A 和版本 B，版本 A 和版本 B 的 GPIO 管脚存在差异；第二，wiringPi 对树莓派的管脚重新进行了封装，例如 wiringPi 的 GPIO0 意味着 BCM 的 GPIO17，这仅仅是一种封装映射关系，不会对开发和使用产生较大的影响。

## 3.3.3. 简单例程

本节讲述如何应用 wiringPi 的 GPIO 功能在树莓派上实现 IO 端口的操作。

### 3.3.3.1. Blink 例程

最简单的使用例子可以参考 wiringPi 官网的 Blink 点亮引脚的实验，官网地址是：http://wiringPi.com/examples/blink/

打开 NotePad++文本编辑器新建一个叫 blink.c 的文本：

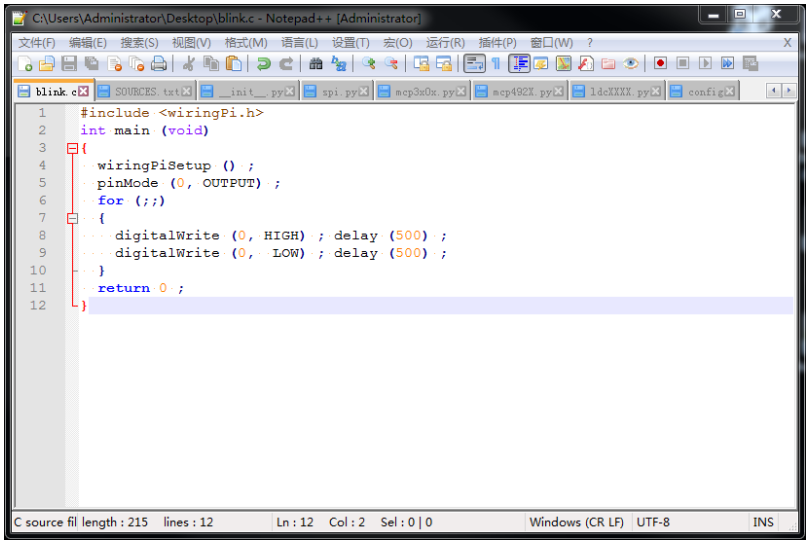



图 3-5 wiringPi 点亮 GPIO 端口源码

需要留意的是树莓派运行的是 Linux 系统文件支持的是 UTF-8 格式并且文档格式也需要更换为 unix 格式。

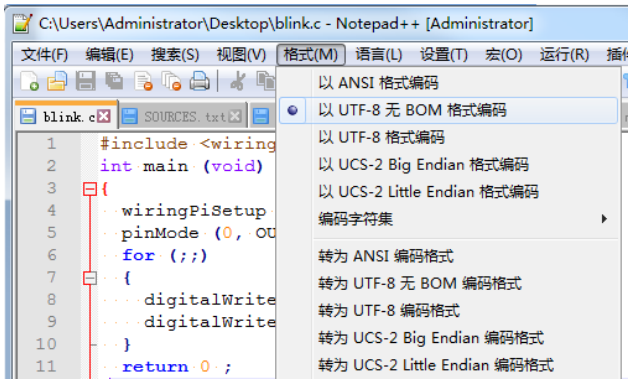



图 3-6 修改文本编码格式

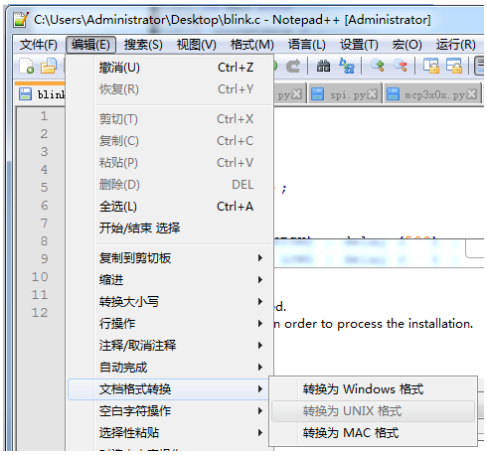



图 3-7 修改文档格式

完成编写后，通过 filezilla 工具上传到树莓派中：如下图所示，在左侧菜单栏中选中 blink.c 文件右键选中上传，通过 ftp 服务会将文件上传到树莓派的 /home/pi 文件夹中。

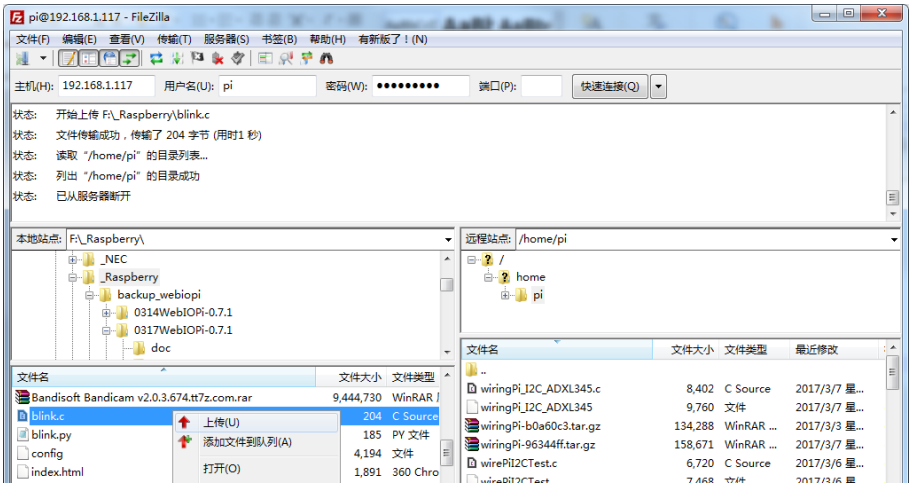



图 3-8 上传源码到 Raspberry

通过树莓派命令行输入：$ls -l 查找/home/pi 下的所有文件可以看到 blink.c 的文件。

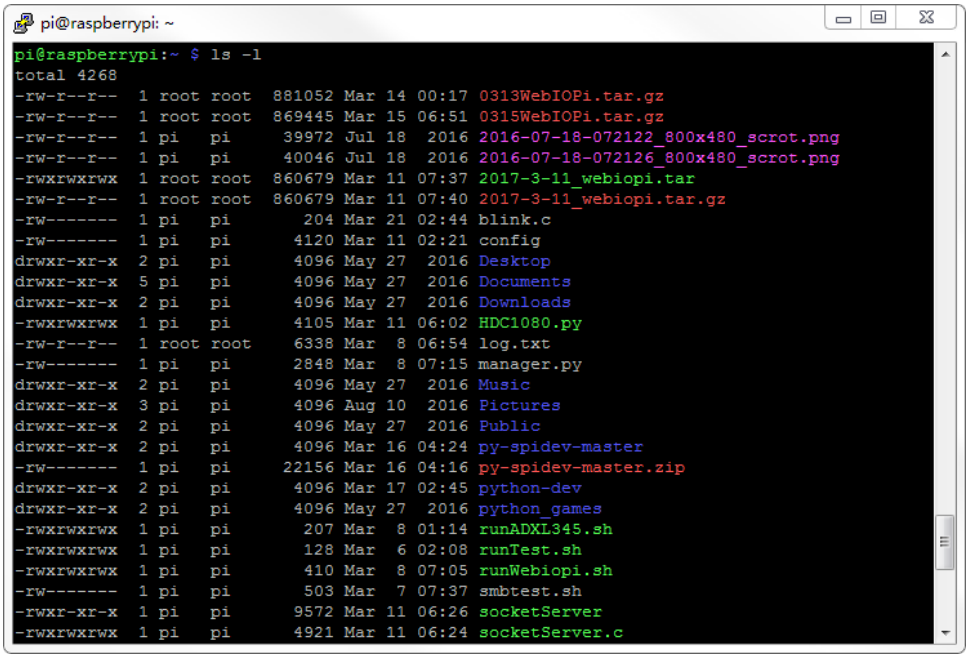



图 3-9 查看 Raspberry 文件系统

完成后通过 gcc 编译该 c 源文件：

```
pi@raspberrypi:~ $ gcc -Wall -o blink blink.c -lwiringPi
```

会生成一个叫 blink 的可执行文件。重新使用 ls -l 再次查找会发现它。

```
pi@raspberrypi:~ $ ls -l
total 4276
-rw-r--r-- 1 root root 881052 Mar 14 00:17 0313WebIOPi.tar.gz
-rw-r--r-- 1 root root 869445 Mar 15 06:51 0315WebIOPi.tar.gz
-rw-r--r-- 1 pi pi 39972 Jul 18 2016 2016-07-18-072122_800x480_scrot.png
-rw-r--r-- 1 pi pi 40046 Jul 18 2016 2016-07-18-072126_800x480_scrot.png
-rwxrwxrwx 1 root root 860679 Mar 11 07:37 2017-3-11_webiopi.tar
-rw-r--r-- 1 root root 860679 Mar 11 07:40 2017-3-11_webiopi.tar.gz
-rwxr-xr-x 1 pi pi 6388 Mar 21 02:49 blink
-rw------- 1 pi pi 204 Mar 21 02:44 blink.c
```

图 3-10 生成可执行文件 blink

在后面需要一个 LED 灯连接到树莓派的输出板上，如下如所示，连接 LED 的正负极到 GPIO17 和 GND 上。其中 GPIO17 对应的就是 wiringPi 的映射引脚 0 口。

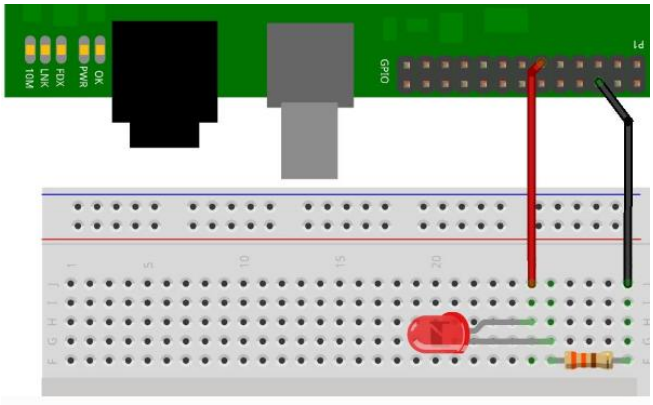



图 3-11 连接 LED 到树莓派

完成了硬件连接后，在命令行输入 sudo ./blink 执行应用程序：

```
^Cpi@raspberrypi:~ $ sudo ./blink
```

可以看到 LED 开始不断地闪烁。可以通过 Ctrl+C 键停止运行程序，小灯停止闪烁。

图 3-9 blink 运行效果

## 3.4. 总结

本章节介绍 wiringPi 的安装和简单实用例程的主要目的是让初学者理解树莓派上的跨平台嵌入式开发的流程，以及如何通过代码控制树莓派的 GPIO 端口。主要目的是为了后续介绍 webiopi 打下一个感性的基础。

# 4. WebIOPI

## 4.1. 简介

webiopi 是一个可控制树莓派 GPIO 的 web 框架，该框架面向物联网 IoT 开发。该项目托管于 google，并可在 sourceforge 上下载到源代码。现在 webiopi 已经发展到 0.7.1 版本，webiopi 支持 REST Server，CoAP server，并提供 python 库和 javascript 库，毫无疑问的说 webiopi 是一个优秀的框架。从 sourceforge 上的信息来看，下载量前三位为美国，德国和英国，中国的下载量排在了 26 位（2014 年 3 月 8 日），所以我觉得应该写点什么普及一下 webiopi，并顺便激励一下仍在玩“单片机”的中国嵌入式工程师们，往前走“一步”海阔天空。

虽然 webiopi 是一个优秀的框架，但是 webiopi 由 python，javacript 语言编写，并提供了 REST 和 CoAP 形式访问方式，学习和掌握 webiopi 需要不少的基础知识。对于那些认为 C 语言、寄存器操作、16 进制协议可以解决一切问题的嵌入式工程师来说，python 和 javacript，REST 和 CoAP 协议显得臃肿不堪并毫无用处。但是事实并非如此。

（引用自：http://blog.csdn.net/xukai871105/article/details/20799551）


图 4-1 webiopi 框架简介

WebIOPi 的特性：

- 用 Python 编写，具有加载和执行自定义脚本的功能，使用具有初始化（setup）和循环（loop）功能的全面结构
- 统一串口/ SPI / I2C 支持与一套完整和一致的功能来控制 30 多个设备，包括

最常用的模拟转换器，I / O 扩展和传感器

- JavaScript / HTML 客户端库制作 web 界面
- Python / java 客户端，支持树莓派之间系统或 Android 应用程序
- COAP 支持带来了最好的物联网协议的 PI，PI 可能作为未来的证明
- 包括简单的 Web 应用程序，调试 GPIO、设备和串行接口

## 4.2. WebIOPi 的安装

### 4.2.1. 下载 WebIOPi

WebIOPi 的程序可以在官方网站 http://webiopi.trouch.com/ 或者通过 sourceforge 下载最新的 WebIOPi 程序。

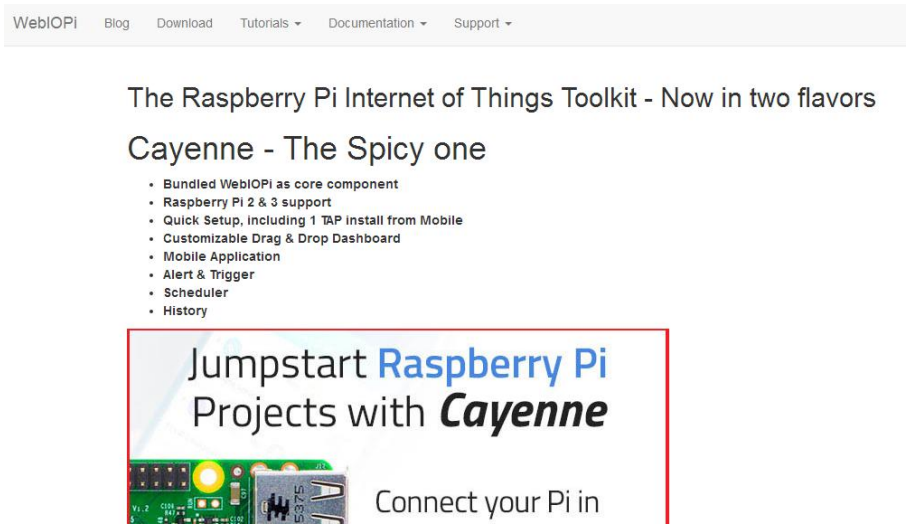



图 4-2 webiopi 官网主页

### 4.2.2. FTP 上传至树莓派

下载完成后再本地 PC 上得到一个 WebIOPi-0.7.1.tar.gz 的压缩包。通过 FTP 服务可以上传到树莓派的/home/pi 文件夹中，操作方法不在此章节中赘述。

### 4.2.3. 解压软件安装

完成上传后能够在/home/pi 上看到响应的文件。

```
-rw-------  1 pi   pi    213894 Mar  9 05:56 WebIOPi-0.7.1.tar.gz
-rwxr-xr-x  1 pi   pi      7468 Mar  6 05:48 wirePiI2CTest
-rw-------  1 pi   pi      6720 Mar  6 05:48 wirePiI2CTest.c
drwxr-xr-x 11 pi   pi      4096 Mar  6 05:19 wiringPi
drwxr-xr-x 10 pi   pi      4096 Mar  7 06:08 wiringpi2
-rw-------  1 pi   pi    158671 Mar  7 05:55 wiringPi-96344ff.tar.gz
-rw-------  1 pi   pi    134288 Mar  3 06:14 wiringPi-b0a60c3.tar.gz
-rwxr-xr-x  1 pi   pi      9760 Mar  7 07:03 wiringPi_I2C_ADXL345
-rw-------  1 pi   pi      8402 Mar  7 07:00 wiringPi_I2C_ADXL345.c
pi@raspberrypi:~ $
```

图 4-3 webiopi 压缩包在 Raspberry 文件系统

然后输入

$ tar xvzf WebIOPi-0.7.1tar.gz 解压安装包到当前文件夹；

$ cd WebIOPi-0.7.1 进入加压后的文件夹

$sudo ./setup.sh 进行程序安装

但是由于树莓派的更新，如果是使用 Raspberry 3 的话扩展引脚的增加会导致运行异常需要给 WebIOPi 打补丁。好在 Google 了半天以后发现 GitHub 上已经有大神完美解决的这个问题，现在附上链接供大家参考。

https://github.com/doublebind/raspi

受益于这个项目可以通过以下步骤完美解决兼容性问题：

$wget

https://raw.githubusercontent.com/doublebind/raspi/master/webiopi-pi2bplus.patch 获取补丁包内容：

```
pi@raspberrypi:~ $ wget https://raw.githubusercontent.com/doublebind/raspi/master/webiopi
-pi2bplus.patch
--2017-03-21 05:14:44--  https://raw.githubusercontent.com/doublebind/raspi/master/webiop
i-pi2bplus.patch
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.100.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.100.133|:443.
.. connected.
HTTP request sent, awaiting response... 200 OK
Length: 5260 (5.1K) [text/plain]
Saving to: 'webiopi-pi2bplus.patch'

webiopi-pi2bplus.patch 100%[=========================>]   5.14K  --.-KB/s   in 0.001s

2017-03-21 05:14:47 (5.08 MB/s) - 'webiopi-pi2bplus.patch' saved [5260/5260]

pi@raspberrypi:~ $
```

图 4-4 webiopi 打补丁 1

$ patch -p1 -i webiopi-pi2bplus.patch 将补丁包内容 patch 的安装程序中

```
pi@raspberrypi:~/WebIOPi-0.7.1 $ patch -p1 -i webiopi-pi2bplus.patch
patching file htdocs/webiopi.js
Reversed (or previously applied) patch detected!  Assume -R? [n] y
patching file python/native/cpuinfo.c
Reversed (or previously applied) patch detected!  Assume -R? [n] y
patching file python/native/gpio.c
patching file python/webiopi/utils/version.py
pi@raspberrypi:~/WebIOPi-0.7.1 $
```

图 4-5 webiopi 打补丁 2

有兴趣的同学可以比较一下对应的修改的几个文件，其实就是为了适应新的通用接口，将原来的 26 引脚接口修改为了 40 引脚的接口。

$ sudo ./setup.sh 重新安装 WebIOPi 程序

测试安装是否成功，可输入以下命令，$ webiopi -h，若安装成功便可获得以

下输出。

```
pi@raspberrypi:~/WebIOPi-0.7.1 $ webiopi -h
WebIOPi command-line usage
webiopi [-h] [-c config] [-l log] [-s script] [-d] [port]

Options:
  -h, --help            Display this help
  -c, --config file     Load config from file
  -l, --log file        Log to file
  -s, --script file     Load script from file
  -d, --debug           Enable DEBUG

Arguments:
  port                  Port to bind the HTTP Server
pi@raspberrypi:~/WebIOPi-0.7.1 $
```

图 4-6 webiopi 成功安装

## 4.2.4. 运行 WebIOPi

完成安装后可以以默认安装的方式运行 WebIOPi。输入如下指令运行 WebIOPi

$ sudo webiopi -d -c /etc/webiopi/config

```
pi@raspberrypi:~/WebIOPi-0.7.1 $ sudo webiopi -d -c /etc/webiopi/config
2017-03-21 05:23:10 - WebIOPi - INFO - Starting WebIOPi/0.7.1/Python3.4
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.digitalCount to REST GET /GPIO/count
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.digitalRead to REST GET /GPIO/%(chan
nel)d/value
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.digitalWrite to REST POST /GPIO/%(ch
annel)d/value/%(value)d
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.getFunctionString to REST GET /GPIO/
%(channel)d/function
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.getPulse to REST GET /GPIO/%(channel
)d/pulse
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.outputSequence to REST POST /GPIO/%(
channel)d/sequence/%(args)s
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.portRead to REST GET /GPIO/*/integer
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.portWrite to REST POST /GPIO/*/integ
er/%(value)d
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.pulse to REST POST /GPIO/%(channel)d
/pulse/
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.pulseAngle to REST POST /GPIO/%(chan
nel)d/pulseAngle/%(value)f
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.pulseRatio to REST POST /GPIO/%(chan
nel)d/pulseRatio/%(value)f
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.setFunctionString to REST POST /GPIO
/%(channel)d/function/%(value)s
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping GPIO.wildcard to REST GET /GPIO/*
2017-03-21 05:23:10 - WebIOPi - INFO - GPIO - Native mapped to REST API /GPIO
2017-03-21 05:23:10 - WebIOPi - INFO - Loading configuration from /etc/webiopi/config
2017-03-21 05:23:10 - WebIOPi - DEBUG - Setup GPIO 21
2017-03-21 05:23:10 - WebIOPi - DEBUG - Setup GPIO 23
2017-03-21 05:23:10 - WebIOPi - DEBUG - Setup GPIO 24
2017-03-21 05:23:10 - WebIOPi - DEBUG - Setup GPIO 25
2017-03-21 05:23:10 - WebIOPi - INFO - Loading SPI modules
2017-03-21 05:23:10 - WebIOPi - DEBUG - Loading module : spi-bcm2708
modprobe: FATAL: Module spi-bcm2708 not found.
2017-03-21 05:23:10 - WebIOPi - DEBUG - Loading module : spidev
2017-03-21 05:23:10 - WebIOPi - DEBUG - Mapping LDC1000(chip=0).getCentimeter to REST GET
```

图 4-7 webiopi 运行后台打印信息

其中：

- -d 代表打开调试模式，运行 webiopi 时会在控制台中输出若干信息。
- -c 代表设置配置文件，配置文件的路径为/etc/webiopi/config，配置文件中有哪些内容，这个会在以后的博客中分析。

若未端口设置 webiopi 的端口，端口号的默认值为 8000

在浏览器（推荐使用 Google Chrome 浏览器）中输入树莓派的 IP 地址和访问端口。192.168.1.107:8000。浏览器提示输入访问身份验证信息，输入用户名为：webiopi，密码：raspberry，回车确认输入。如下如所示：

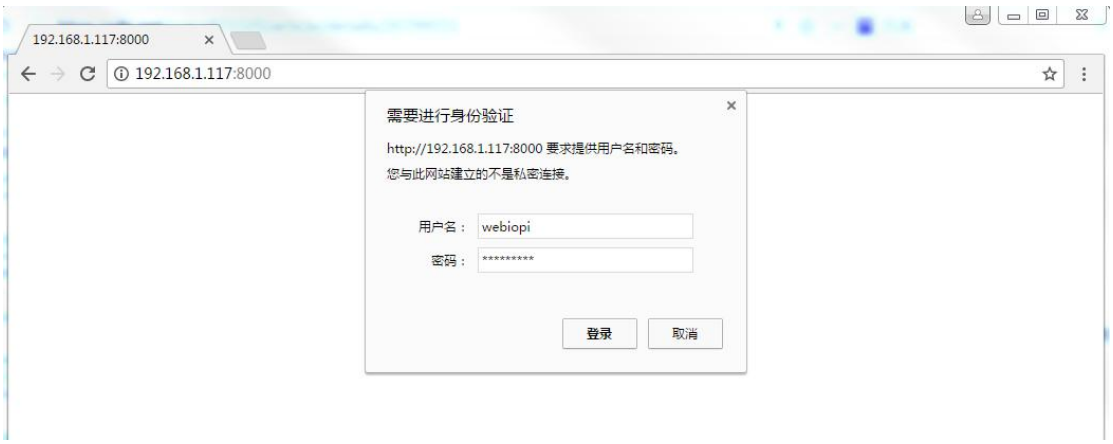



图 4-8 webiopi 网页访问登陆

而后就会进入 webiopi 的网页主界面了。

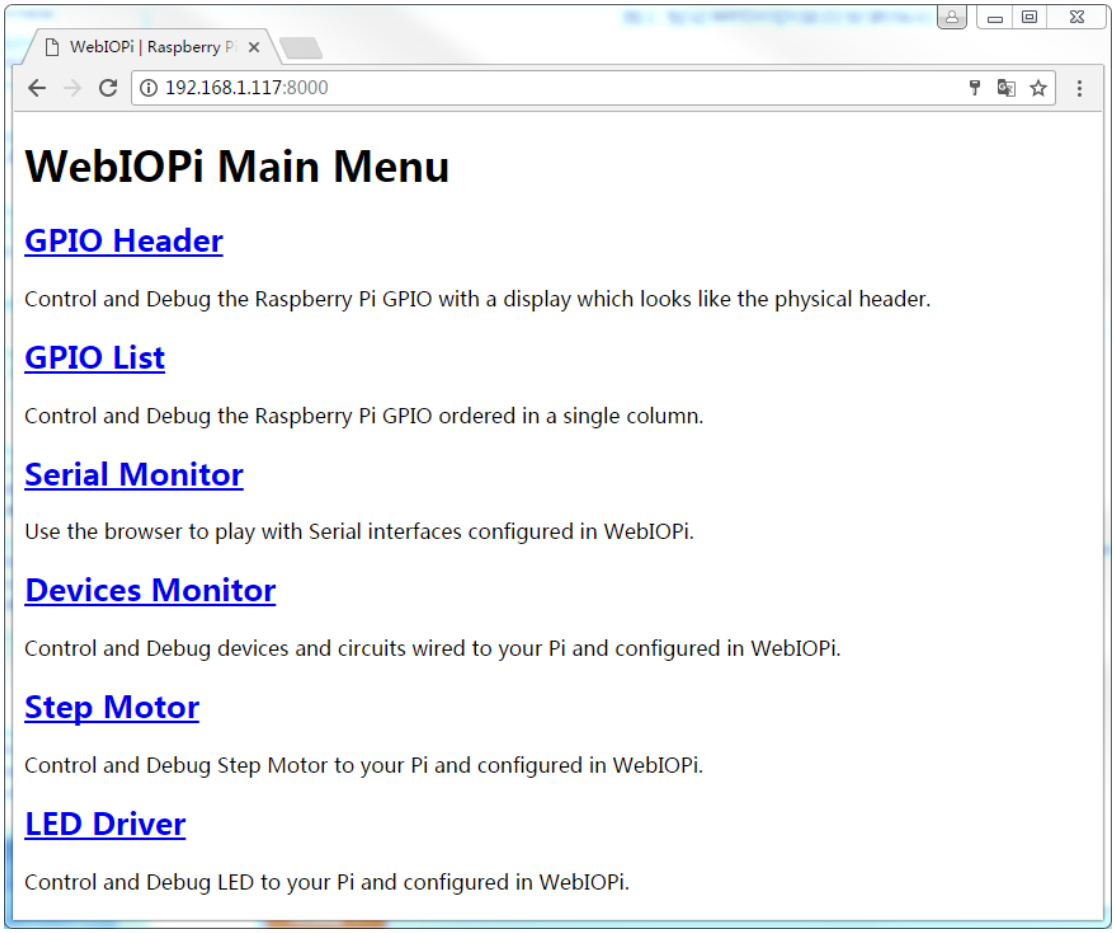



图 4-9 webiopi 主页面

点击进入 GPIOHeader 链接，显示如下：

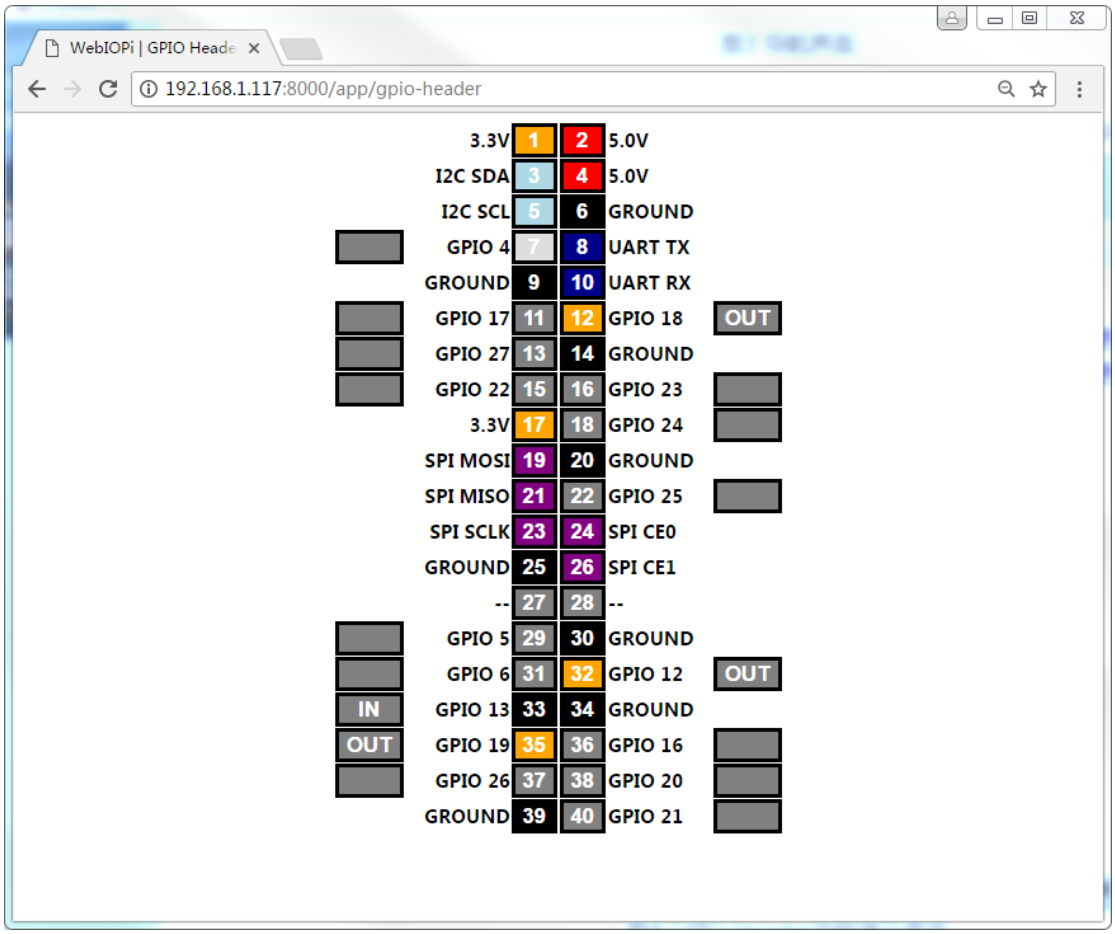



图 4-10 webiopi GPIO 端口列表

其中橙色端口表示打开了和使能的端口。灰色按钮表示通用的 GPIO 端口，显示状态表示已经使能，如果没有显示表示未做初始化。点击有文字的按钮可以修改已经使能端口的状态，如 IN 的话表示 GPIO13 口是输入状态口可以监控端口的高低电平；GPIO18 口是 OUT 模式，则表示可以随意修改端口的输出高低电平状态，假如该端口链接来 LED 灯的话就可以像开关灯泡一样鼠标直接点击端口号改变端口输出高低电平状态了。

到目前这一步表示 webiopi 的初始配置已经完成了。

# 4.3. 背景知识

## 4.3.1. REST

### 4.3.1.1. 简介

REST -- REpresentational State Transfer 直接翻译：表现层状态转移。首先，之

所以晦涩是因为前面主语被去掉了，全称是 Resource Representational State Transfer：通俗来讲就是：资源在网络中以某种表现形式进行状态转移。分解开来：

- Resource：资源，即数据（前面说过网络的核心）。比如 newsfeed，friends 等；
- Representational：某种表现形式，比如用 JSON，XML，JPEG 等；
- State Transfer：状态变化。通过 HTTP 动词实现。

**简单的理解就是：**

**URL 定位资源，用 HTTP 动词（GET,POST,DELETE,DETC）描述操作。**

一、REST 描述的是在网络中 client 和 server 的一种交互形式；REST 本身不实用，实用的是如何设计 RESTful API（REST 风格的网络接口）；

二、Server 提供的 RESTful API 中，URL 中只使用名词来指定资源，原则上不使用动词。“资源”是 REST 架构或者说整个网络处理的核心。比如：

http://api.qc.com/v1/newsfeed：获取某人的新鲜;

http://api.qc.com/v1/friends：获取某人的好友列表;

http://api.qc.com/v1/profile：获取某人的详细信息;3. 用 HTTP 协议里的动词来实现资源的添加，修改，删除等操作。即通过 HTTP 动词来实现资源的状态扭转：

GET 用来获取资源，

POST 用来新建资源（也可以用于更新资源），

PUT 用来更新资源，

DELETE 用来删除资源。比如：

DELETE http://api.qc.com/v1/friends：删除某人的好友（在 http parameter 指定好友 id）

POST http://api.qc.com/v1/friends：添加好友

UPDATE http://api.qc.com/v1/profile：更新个人资料

禁止使用： GET http://api.qc.com/v1/deleteFriend

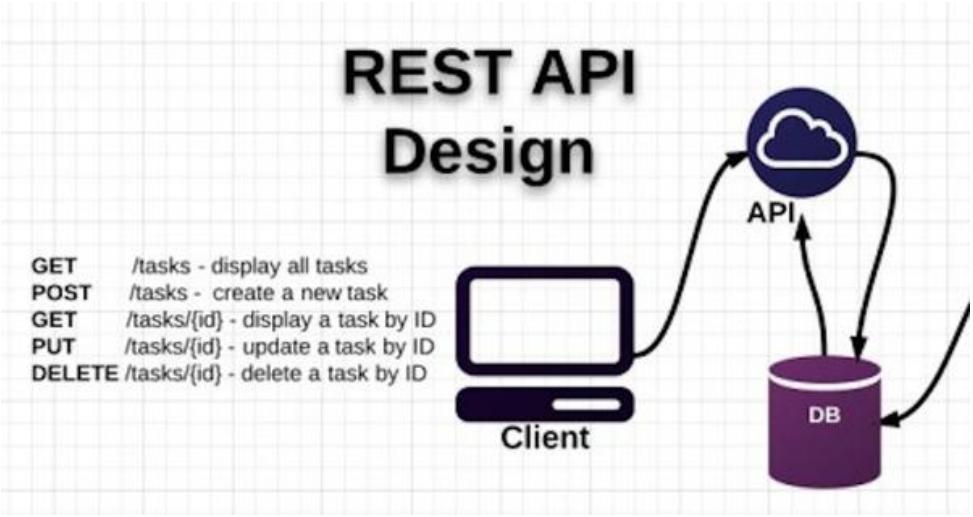



图 4-11 REST 架构

三、Server 和 Client 之间传递某资源的一个表现形式，比如用 JSON，XML 传输文本，或者用 JPG，WebP 传输图片等。当然还可以压缩 HTTP 传输时的数据（on-wire data compression）。

四、用 HTTP Status Code 传递 Server 的状态信息。比如最常用的 200 表示成功，500 表示 Server 内部错误等。

主要信息就这么点。最后是要解放思想，Web 端不再用之前典型的 PHP 或 JSP 架构，而是改为前段渲染和附带处理简单的商务逻辑（比如 AngularJS 或者 BackBone 的一些样例）。Web 端和 Server 只使用上述定义的 API 来传递数据和改变数据状态。格式一般是 JSON。iOS 和 Android 同理可得。由此可见，Web，iOS，Android 和第三方开发者变为平等的角色通过一套 API 来共同消费 Server 提供的服务。

对于习惯来嵌入式开发和 C 语言描述的工程师来说，这些名词和概念真的就像天书一样完全不能理解。但是要想玩转 WebIOPi 就必须了解这些基本概念。

链接：https://www.zhihu.com/question/28557115/answer/78766592

**一句话，REST 本质上是一种分布式系统的应用层解决方案。**

**传统的分布式系统试图在操作系统这一层对底层硬件、软件和数据进行透明管理，向用户虚拟化出一台计算机，用户并不知道底层硬件、软件和数据的具体情形。**

众所周知，计算机系统的系统结构包括底层硬件、操作系统、系统支撑软件、应用层。操作系统占据了重要的地位，提供了两个最根本功能，资源管理与接口。

其中，资源管理，包括硬件资源和软件资源。接口包括编程接口和人机交互接口。当计算机逐渐普及后，部署了越来越多的计算机系统。人们试图将这些计算机系统整合起来。

90 年代，改造通用的单机操作系统，将分布在地理位置不同的软、硬件资源管理起来，透明地提供给用户。很可惜，问题很多，通用的分布式系统的解决方案是很困难的。而且人们对于这种分布式系统的需求也并不是那么迫切。

当 SNS 兴起之后，用户的信息分散在不同的平台上，用户所需要的信息等资源也分散在不同的平台上。传统的分布式系统，改造的是操作系统。但是，显然包括操作系统在内的软、硬件都属于不同的公司。

怎么办？

一个简单的思想，将资源的互通与资源的管理分离开。就是在应用层去满足这个资源互联互通的需求。信息等软、硬件资源还是由各个公司自己在自己的服务器上去解决，这就不需要资源管理了，也就不需要象传统的分布式系统那样需要改造操作系统的资源管理方案。

另外一方面，http 位于网络的应用层，soa 的思想也比较成熟，能够很好地满足资源互通的需求，因此，只需设计一个应用层的资源互联互通的协议就可以了。至于各种资源的管理，还是丢给各个公司自己的操作系统去管理好了。

这就把资源互通与资源管理这两个问题分离开了。

搞懂这个，再去看 rest 是如何在应用层通过 http 实现资源的互联互通，就易如反掌。

本节引用自 https://www.zhihu.com/question/28557115/answer/48094438

### 4.3.2. RESTful

REST， 名词，一种网络架构规范。

RESTful，形容词，指实现了 REST 规范的系统，如实现了 REST 规范的 Web API 就叫 RESTful API。

为了理解 RESTful 我们举个例子：

（链接：https://www.zhihu.com/question/28557115/answer/146660220）

首先我们假设西雅图有个男孩叫小明，他是一个科比的粉丝，然后有一天小明想用电脑看科比的生涯集锦视频。他做了如下四步：


图 4-12 常用访问步骤

在这个活动中，主要涉及到了四个部件：


图 4-14 常用访问步骤构成

这大概就是一个传统的电脑软件的架构。

现在我们看看如何把这个传统的电脑软件变成 RESTful 架构的软件。

我们把小明家的电脑主机从小明西雅图搬到加州的圣布鲁罗，而小明的显示器则仍然留在家里。

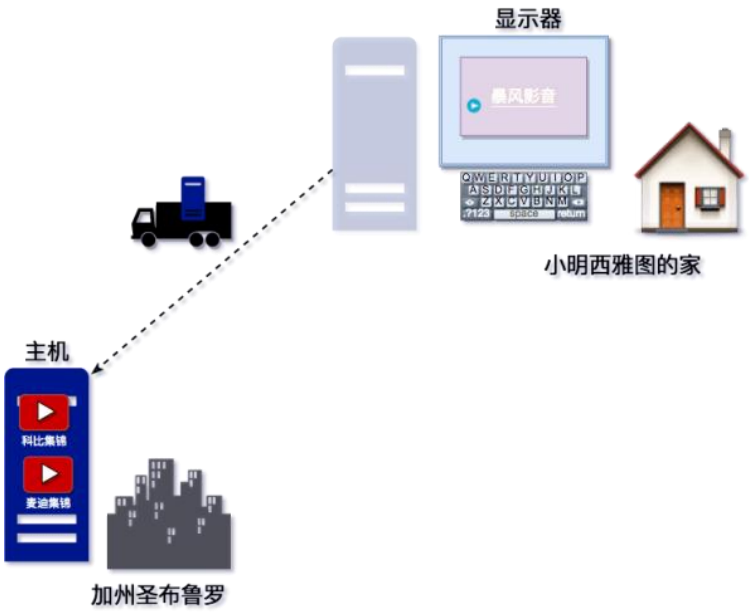



图 4-15 移动主机改为网络访问

然后我们把之前连接电脑主机和显示器的“电线”换成“互联网”，并且把四个部件（显示器，主机，视频文件，暴风影音软件）的名字换成：客户端，服务器，资源，浏览器。

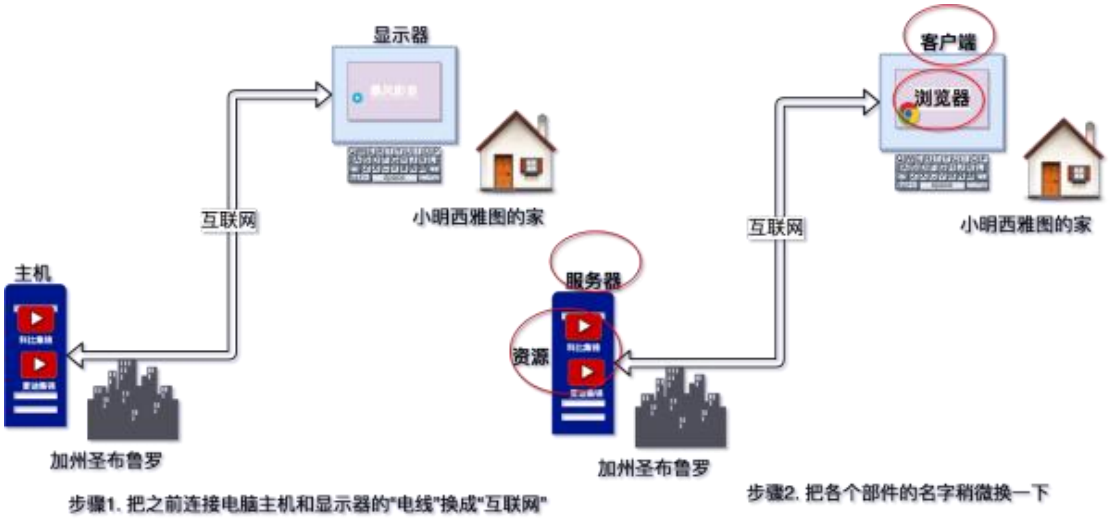



图 4-16 更改访问模式

为了在茫茫互联网世界中准确快速的找到那台服务器并观看“科比集锦”，我们需要给在加州圣布鲁罗的这台服务器加一个标识符，也就是常说的 Url。同时，为了区分服务器中的“科比集锦”和“麦迪集锦”，我们也需要给这两个视频单独加上标识符。

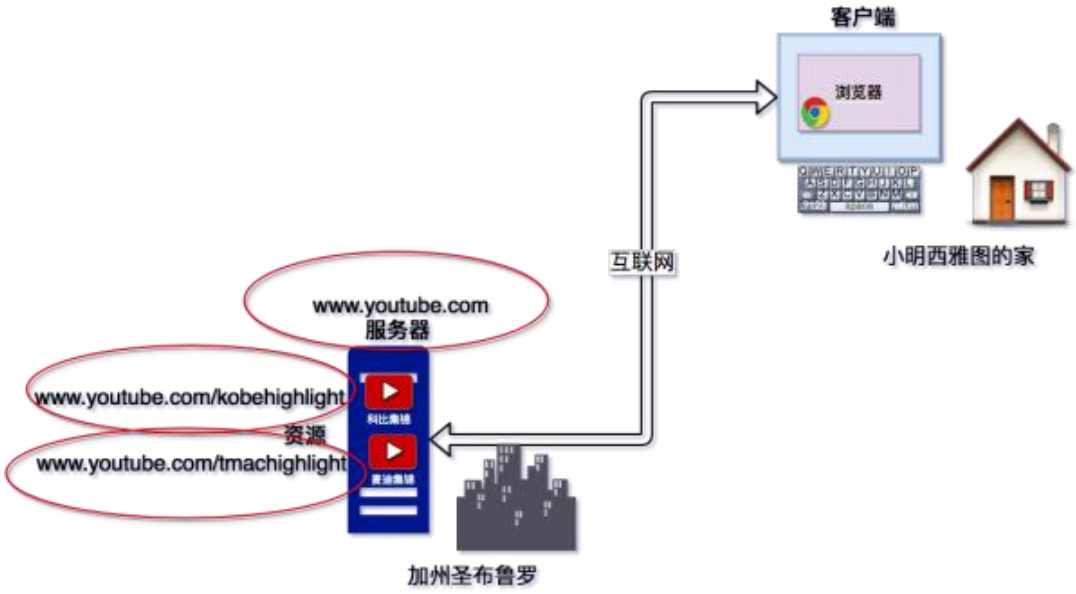



图 4-17 添加 URL 标示

这个服务器当然不只有科比和麦迪的视频，还有其他千千万万的视频，以及音乐，表单和网页等等东西。服务器上放不下这么多东西，所以需要把这些东西移到数据库里面放着。

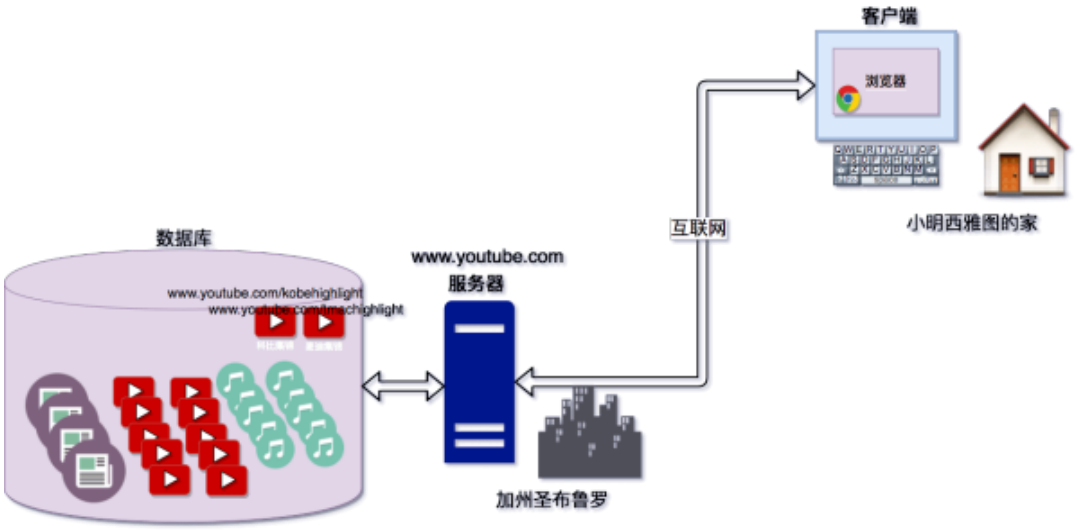



图 4-18 建立数据库访问模式

这样，整个 RESTFul 的架构基本上就成型了。我们再来看看小明今天应该如何观看科比的视频。

1. 小明打开浏览器。注意此时小明家中的电脑硬盘里面没有存任何视频，视频存在加州圣布鲁罗的数据库里面。

2.

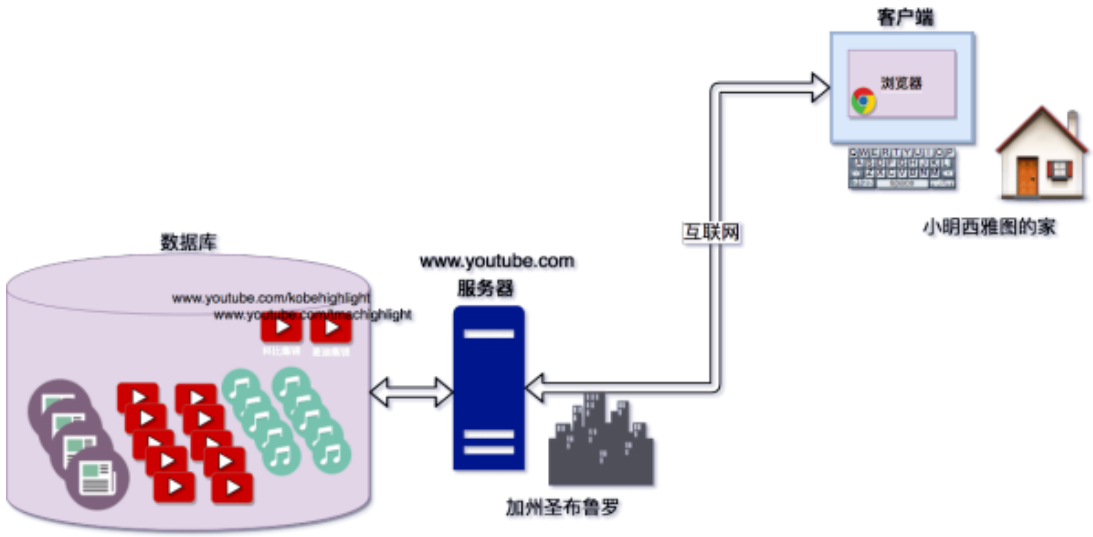



图 4-19 RESTful 架构

小明输入 http://www.youtube.com/kobehighlight 客户端通过互联网找到 http://www.youtube.com 的服务器，然后服务器根据 kobehighlight 在数据库里找到了科比的视频，并把视频数据通过互联网传回给了小明的客户端。这个操作就是我们常说的 GET。

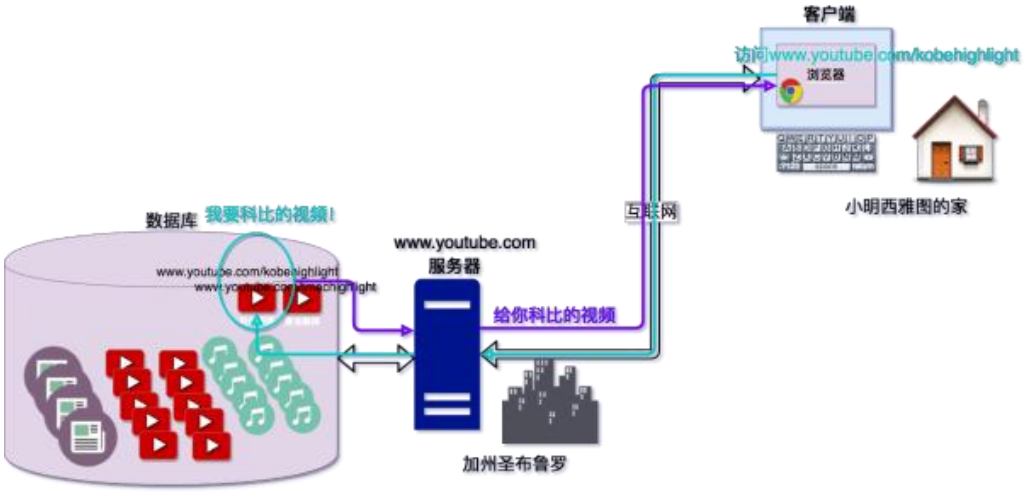



图 4-20 RESTful 访问流程

3. 小明开心地观看传回来的视频，但是他不想在视频中看到克雷汤普森。

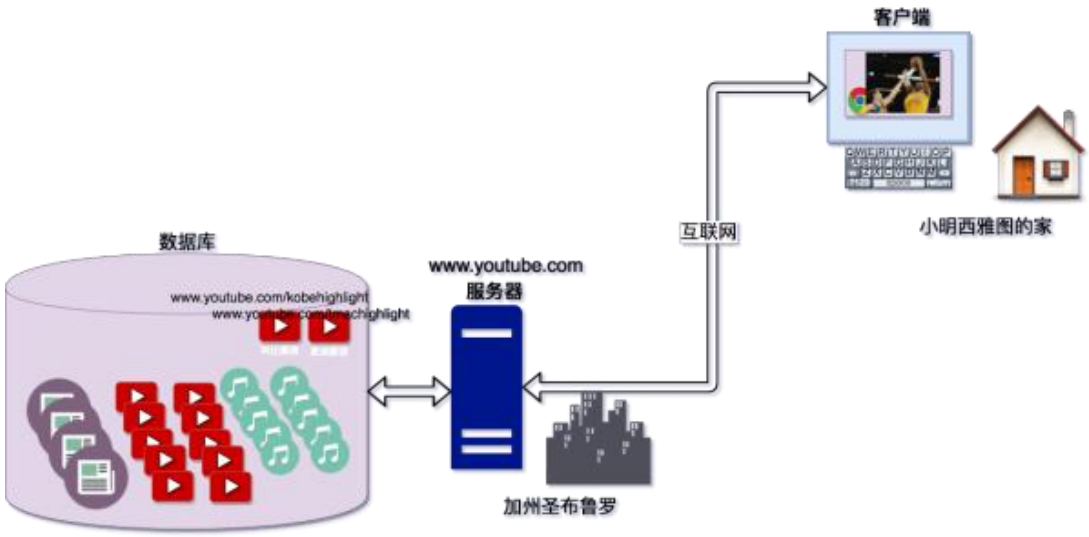



图 4-21 RESTful get 数据

4. 他通过浏览器剪掉了克雷汤普森的部分，点击了”提交“，然后服务器接到这个请求之后把修改保存到数据库里面，并且告诉小明”你的修改已经保存。这个操作就是我们常说的 POST。

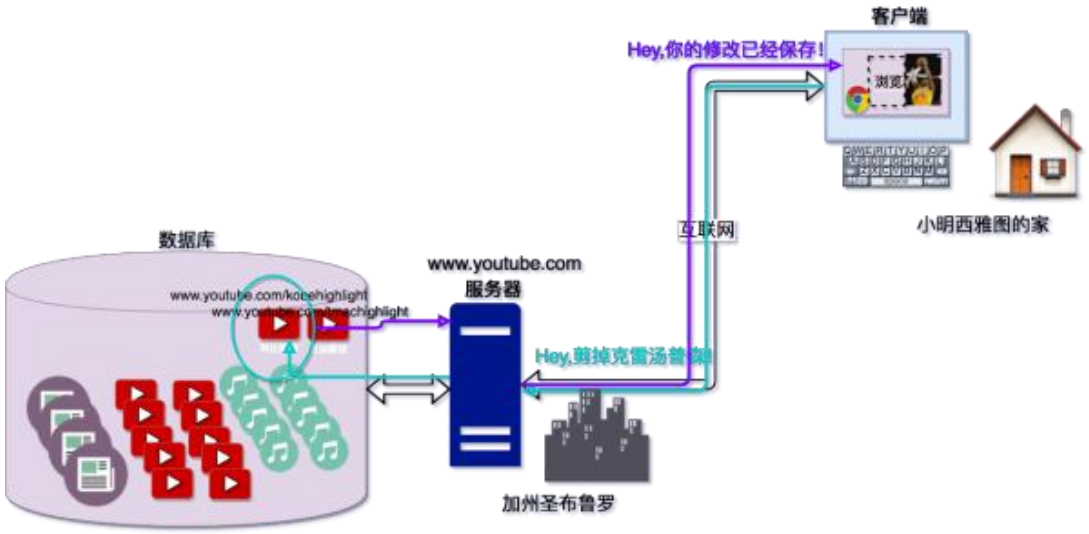



图 4-22 RESTful post 数据

我们来对比一下之前的单机软件。

- 单机软件的文件路径 = REST 软件的 Url
- 单机软件的打开操作 = REST 软件 HTTP 的 GET
- 单机软件的保存操作 = REST 软件 HTTP 的 POST

当然，其实还有很多细节，不过这就是 REST 软件框架的一个大概样子。

## 4.3.3. COAP 协议

### 4.3.3.1. CoAP 简介

CoAP 是受限制的应用协议(Constrained Application Protocol)的代名词。最近几年专家们预测会有更多的设备相互连接，而这些设备的数量将远超人类的数量。在这种大背景下，物联网和 M2M 技术应运而生。虽然对人们而言，连接入互联网显得方便容易，但是对于那些微型设备而言接入互联网非常困难。在当前由 PC 机组成的世界，信息交换是通过 TCP 和应用层协议 HTTP 实现的。但是对于小型设备而言，实现 TCP 和 HTTP 协议显然是一个过分的要求。为了让小设备可以接入互联网，CoAP 协议被设计出来。CoAP 是一种应用层协议，它运行于 UDP 协议之上而不是像 HTTP 那样运行于 TCP 之上。CoAP 协议非常小巧，最小的数据包仅为 4 字节。

### 4.3.3.2. CoAP 协议是否可以替换 HTTP 协议？

CoAP 并不能替代 HTTP 协议，但是对于那些小设备（256KB Flash 32KB RAM 20MHz 主频）而言 CoAP 的确是一个好的解决方案。

### 4.3.3.3. CoAP 消息类型

CoAP 采用与 HTTP 协议相同的请求响应工作模式。CoAP 协议共有 4 中不同的消息类型。

- CON——需要被确认的请求，如果 CON 请求被发送，那么对方必须做出响应。
- NON——不需要被确认的请求，如果 NON 请求被发送，那么对方不必做出回应。
- ACK——应答消息，接受到 CON 消息的响应。
- RST——复位消息，当接收者接受到的消息包含一个错误，接受者解析消息或者不再关心发送者发送的内容，那么复位消息将会被发送。

### 4.3.3.4. CoAP 消息结构

一个 CoAP 消息最小为 4 个字节，以下是 CoAP 协议不同部分的描述。

- 【版本 Version】：类似于 IPv6 和 IPv6，仅仅是一个版本号。
- 【消息类型 Message Type】：CON，NON，ACK，RST。

- 【消息 ID Message ID】：每个 CoAP 消息都有一个 ID，在一次会话中 ID 总是保持不变。但在这个会话之后该 ID 会被回收利用。
- 【标记 Token】：标记是 ID 的另一种表现。
- 【选项 Options】：CoAP 选项类似于 HTTP 请求头，它包括 CoAP 消息本身，例如 CoAP 端口号，CoAP 主机和 CoAP 查询字符串等。
- 【负载 Payload】：真正有用的被交互的数据。

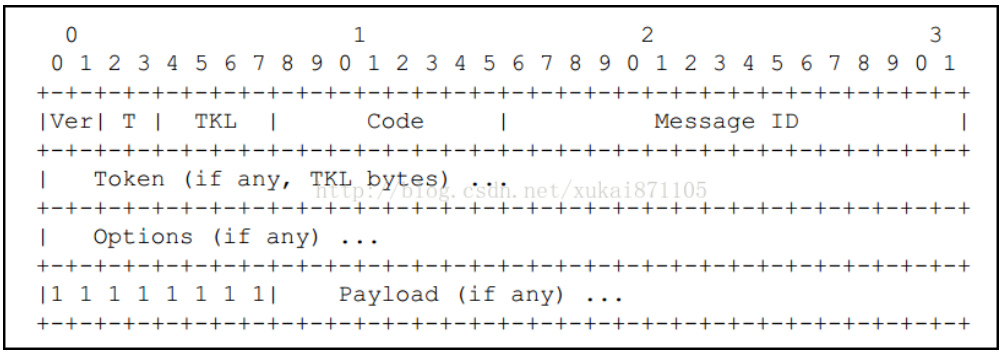



图 4-23 CoAP 消息结构

#### 4.3.3.5. CoAP 的 URL

在 HTTP 的世界中，RESTFul 协议由于其简单性和适用性，在 WEB 应用中越来越受欢迎，这样的道理同样适用于 CoAP。一个 CoAP 资源可以被一个 URI 所描述，例如一个设备可以测量温度，那么这个温度传感器的 URI 被描述为：CoAP://machine.address:5683/sensors/temperature。请注意，CoAP 的默认 UDP 端口号为 5683。

#### 4.3.3.6. CoAP 观察模式

在物联网的世界中，你需要去监控某个传感器例如温度或湿度等。在这种情况下，CoAP 客户端并不需要不停的查询 CoAP 服务器端的数据变化情况。CoAP 客户端可以发送一个观察请求到服务器端。从该时间点开始计算，服务器便会记住客户端的连接信息，一旦温度发生变化，服务器将会把新结果发送给客户端。如果客户端不在希望获得温度检测结果，那么客户端将会发送一个 RST 复位请求，此时服务器便会清除与客户端的连接信息。

#### 4.3.3.7. CoAP 块传输

CoAP 协议的特点是传输的内容小巧精简，但是在某些情况下不得不传输较大的数据。在这种情况下可以使用 CoAP 协议中的某个选项设定分块传输的大小，那么无论是服务器或客户端可完成分片和组装这两个动作。

（本章节应用自 http://blog.csdn.net/xukai871105/article/details/17734163）

## 4.4. Webiopi 运行分析

要研究 Webiopi 的运行机制首先需要了解它的基本构成，为了方便分析可以将解压完成后的整个 WebIOPi-0.7.1 文件夹下载到 PC 端分析它的基本构成；

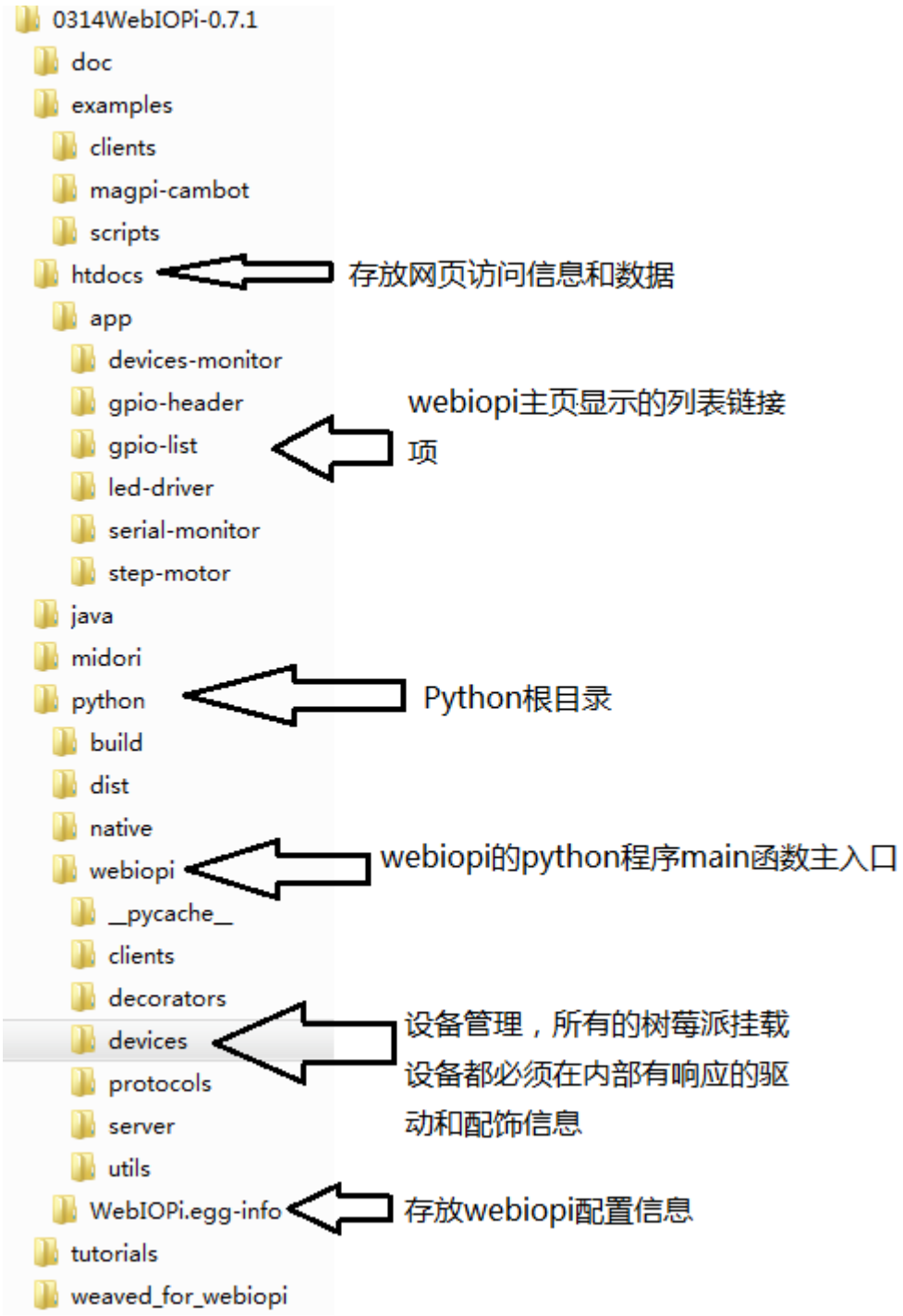



图 4-24 webiopi 文件系统基本构成

## 4.4.1. 配置文件

要熟练使用 webiopi，先要对 webiopi 的配置有所了解，webiopi 通过某一个 config 文件实现所有的配置信息，存放位置位于 WebIOPi-0.7.1\python\config。最终会在安装后将配置拷贝到 Raspberry 文件系统下的/etc/webiopi/config。

#### 4.4.1.1. GPIO 初始化

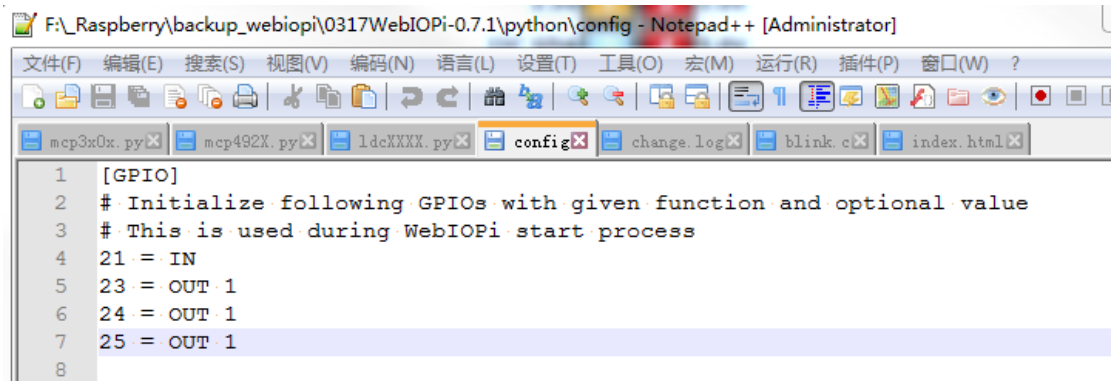



图 4-25 config 配置 GPIO

webiopi 运行之后立即设置 GPIO 端口的状态，例如输入还是输出，若是输出的话输出高电平还是低电平

#### 4.4.1.2. GPIO 重置

```
9  #------------------------------------------------------------------------#
10
11 [~GPIO]
12 # Reset following GPIOs with given function and optional value
13 # This is used at the end of WebIOPi stop process
14 #21 = IN
15 #23 = IN
16 #24 = IN
17 #25 = OUT 0
18
19 #------------------------------------------------------------------------#
```

图 4-26 config 配置 GPIO

webiopi 结束运行之前，设置 GPIO 状态。该设置最好和 GPIO 初始化相对应，更确切的说恢复所有的 GPIO 为输入。

#### 4.4.1.3. 脚本载入

```
19 #------------------------------------------------------------------------#
20
21 [SCRIPTS]
22 # Load custom scripts syntax :
23 # name = sourcefile
24 #    each sourcefile may have setup, loop and destroy functions and macros
25 #myscript = /home/pi/webiopi/examples/scripts/macros/script.py
26 #myscript = /home/pi/WebIOPi-0.7.1/htdocs/app/led-driver/script.py
27 #------------------------------------------------------------------------#
28
```

图 4-27 config 配置 script.py

script.py 可理解为一个主文件，主文件中包含 3 大块内容，setup loop 和 destroy。webiopi 运行之后的顺序依次是：setup 运行一次，webiopi 运行时 loop 连续运行，webiopi 运行结束之前 destroy 运行一次。script.py 文件的主要功能存在于 loop 中，在没有网页操作时，loop 中的相关操作也会运行。

#### 4.4.1.4. HTTP 服务

```
#-------------------------------------------------------------------------#

[HTTP]
# HTTP Server configuration
enabled = true
port = 8000

# File containing sha256(base64("user:password"))
# Use webiopi-passwd command to generate it
passwd-file = /etc/webiopi/passwd

# Change login prompt message
prompt = "WebIOPi"

# Use doc-root to change default HTML and resource files location
#doc-root = /home/pi/webiopi/examples/scripts/macros

# Use welcome-file to change the default "Welcome" file
#welcome-file = index.html

#-------------------------------------------------------------------------#
```

图 4-28 config 配置 http

- 使能或者禁止 HTTP 服务
- 设置 HTTP 服务的端口号，默认为 8000，注意由于存在 web 各种 web 服务器（apache 或者 lighttpd），所有请勿使用 80 端口。
- 设置登录用户名和密码，若是新手建议不用修改。
- 设置 html 文件目录（请注意是目录而不是文件）。该设置经常会被修改。

#### 4.4.1.5. CoAP 服务

```
#-------------------------------------------------------------------------#

[COAP]
# CoAP Server configuration
enabled = true
port = 5683
# Enable CoAP multicast
multicast = true

#-------------------------------------------------------------------------#
```

图 4-29 config 配置 CoAP

- 使能 CoAP 服务
- CoAP 的默认端口为 5683，不建议修改。

### 4.4.1.6. 载入设备

```
55
56  #------------------------------------------------------------------------#
57
58  [DEVICES]
59  # Device configuration syntax:
60  # name = device [args...]
61  #    name   : used in the URL mapping
62  #    device : device name
63  #    args   : (optional) see device driver doc
64  # If enabled, devices configured here are mapped on REST API /device/name
65  # Devices are also accessible in custom scripts using deviceInstance(name)
66  # See device driver doc for methods and URI scheme available
67
68  # Raspberry native UART on GPIO, uncomment to enable
69  # Don't forget to remove console on ttyAMA0 in /boot/cmdline.txt
70  # And also disable getty on ttyAMA0 in /etc/inittab
71  #serial0 = Serial device:ttyAMA0 baudrate:9600
72
73  # USB serial adapters
74  #usb0 = Serial device:ttyUSB0 baudrate:9600
75  #usb1 = Serial device:ttyACM0 baudrate:9600
76
77
78  #temp1 = TMP102 slave:0x53
79  #temp2 = DS18B20
80  #temp3 = DS18B20 slave:28-0000049bc218
81
82  #AD ADX1100转换
83  #temp0 = TMP102
84
85  #加速度传感器
86  #adxl = ADXL345
87
88  #光通量传感器
89  #opt = OPT3001
90
91  #温湿度传感器
92  #hdc = HDC1080
93
94  #接近传感器
95  ldc = LDC1000
```

```
8
9   #gpio0 = PCF8574
0   #gpio1 = PCF8574 slave:0x21
1
2   #light0 = TSL2561T
3   #light1 = TSL2561T slave:0b0101001
4
5   #gpio0 = MCP23017
6   #gpio1 = MCP23017 slave:0x21
7   #gpio2 = MCP23017 slave:0x22
8
9   #pwm0 = PCA9685
0   #pwm1 = PCA9685 slave:0x41
1
2   #adc0 = MCP3008
3   #adc1 = MCP3008 chip:1 vref:5
4   #dac1 = MCP4922 chip:1
5
```

图 4-30 config 配置多种传感器

载入各种设备，由于设备暂缺未详细测试。在接下来的介绍中我们会逐个介绍上图中出现的例如：

- ADXL345 三轴加速度传感器

- ADS1100 AD 转换采集
- OPT3001 光通量传感器
- HDC1080 温湿度传感器
- LDC1000 接近传感器

#### 4.4.1.7. REST 设置

```
[REST]
# By default, REST API allows to GET/POST on all GPIOs
# Use gpio-export to limit GPIO available through REST API
gpio-export = 12, 13, 18, 19

# Uncomment to forbid changing GPIO values
gpio-post-value = true

# Uncomment to forbid changing GPIO functions
gpio-post-function = true

# Uncomment to disable automatic device mapping
#device-mapping = false
```

图 4-31 config 配置 REST

默认的情况下，通过 REST API 可以控制所有的 GPIO 端口。这些危险操作会和 loop 中的功能冲突，为了防止该情况可通过配置限制 REST 的功能，例如设置 GPIO 的方法和输出电平。根据上面介绍的 WebIOPi 默认运行情况可以看到，使能的正好是这个配置中出现的 GPIO 12、GPIO 13、GPIO 18、GPIO 19 四个引脚。

#### 4.4.1.8. URL 重路由

```
[ROUTES]
# Custom REST API route syntax :
# source = destination
#   source      : URL to route
#   destination : Resulting URL
# Adding routes allows to simplify access with Human comprehensive URLs

# In the next example with have the bedroom light connected to GPIO 25
# and a temperature sensor named temp2, defined in [DEVICES] section
#  - GET  /bedroom/light        => GET  /GPIO/25/value, returns the light stat
#  - POST /bedroom/light/0      => POST /GPIO/25/value/0, turn off the light
#  - POST /bedroom/light/1      => POST /GPIO/25/value/1, turn on the light
#  - GET  /bedroom/temperature => GET  /devices/temp2/temperature/c, returns
the temperature in celsius
#/bedroom/light = /GPIO/25/value
#/bedroom/temperature = /devices/temp2/temperature/c

#/livingroom/light = /devices/expander0/0
#/livingroom/brightness = /devices/adc/0/float
#/livingroom/temperature = /devices/temp0/temperature/c

#/weather/temperature = /devices/bmp/temperature/c
```

图 4-32 config 配置 URL

使得 REST API 看起来更优美一些，暂时未详细测试。

## 4.5. GPIO 实现

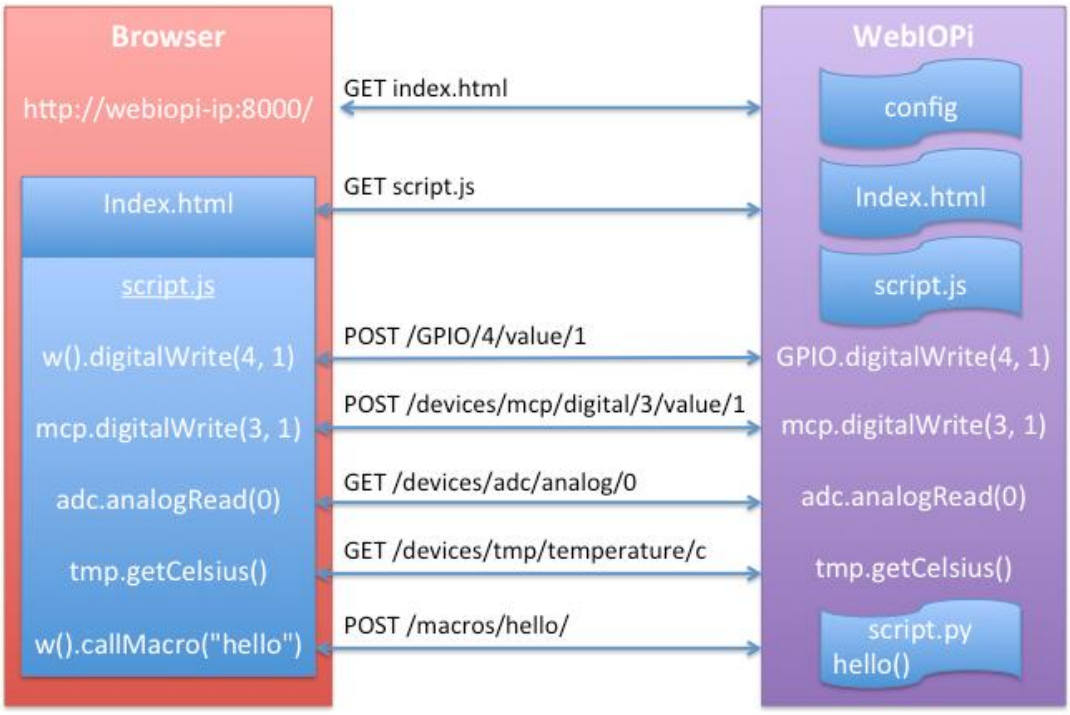



图 4-33 webiopi GPIO 控制流程框架

用 WebIOPi 的物联网应用非常方便，一旦我们抓住了它是如何工作的，什么是必要的，那么做二次开发就容易得多。

webiopi 包括一个 HTTP 服务器，提供 HTML 资源和 REST API 来控制的东西。您的浏览器将首先加载一个 HTML 文件，然后所包含的 JavaScript 将异步调用其他 API 来控制和更新用户界面。这种方法非常有效，因为它不需要刷新和下载整个页面。

- 你可以从零开始建立你的网页的用户界面，使用 webiopi .js 库。
- 你可以通过使用一个像 Arduino 那样初始化/循环功能语法加载自定义 Python 脚本扩展 webiopi 行为。

假设我们只需要一个按钮来控制一个 LED 灯，这将自动打开和关闭。它可以是任何你想要的。定制过程有 3 个步骤：

1. 写一个简单的 Python 脚本来初始化 GPIO 和处理/自动关闭
2. 编写一个简单的 HTML / JavaScript 页面
3. webiopi 服务器配置

## 4.5.1. 准备工作

首先我们在 WebIOPi 的初始化配置基础上做功能添加：

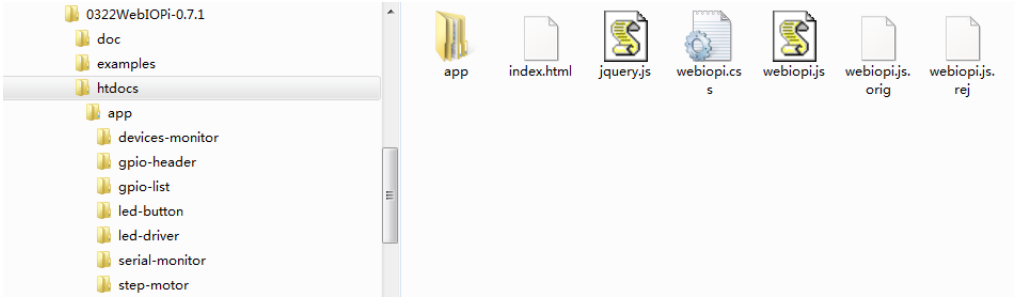



图 4-34 webiopi 网页布局

Htdocs 中存放的是网页访问文件信息，其中/htdocs 文件夹下的 index.html 中可以添加一项 LED Button 链接项：

```
<h2><a href="/app/led-button">LED Button</a></h2>
<p>Control and Debug LED Light button on your Pi</p>
```

并且创建一个叫 led-button 的文件夹，其中创建并存放一个 index.html 和 script.py 文件。

## 4.5.2. 编写 Python 脚本

完成基本配置准备后，根据上面的介绍我们一步步搭建运行环境，首先需要编写一个 script.py 的 Python 脚本来初始化 GPIO 和处理/自动关闭配置。

脚本如下所示：

```
import webiopi
import datetime

# Enable debug output
webiopi.setDebug()

# GPIO Init
GPIO = webiopi.GPIO

# GPIO pin using BCM numbering, config LED port
LIGHT = 17

# setup function is automatically called at WebIOPi startup
def setup():
    # set the GPIO used by the light to output
    GPIO.setFunction(LIGHT, GPIO.OUT)

# loop function is repeatedly called by WebIOPi
def loop():
    webiopi.sleep(1)

# destroy function is called at WebIOPi shutdown
def destroy():
    GPIO.digitalWrite(LIGHT, GPIO.LOW)
```

图 4-35 script.py 初始化端口 GPIO

可以看到整个脚本分为三个部分：

Setup 表示初始化 17 端口为 LED 驱动端口；

Loop 表示循环运行，没有操作仅仅做一个延时一秒操作；

Destroy 表示退出，将 17 端口置低操作。

脚本文件会在 Webiopi 服务访问该页面时自动装载并在后台运行。

```
# setup function is automatically called at WebIOPi startup
defsetup():
# set the GPIO used by the light to output
    GPIO.setFunction(LIGHT, GPIO.OUT)
```

setFunction 为 webiopi 中 javascritp 部分提供的 API 函数，该函数利用 jquery 中的$.post 方法调用 REST API。实现 setFunction 部分的代码——webiopi.js：

```
1. GPIOPort.prototype.setFunction = function(channel, func, callback) {
2.  var name = this.name;
3.  $.post(this.url + "/" + channel + "/function/" + func, function(data) {
4.   callback(name, channel, data);
5.  });
6. }
```

访问 REST API 为 /17/function/out。意为 GPIO17 端口为输出状态。

### 4.5.3. 编写 HTML / JavaScript 页面

创建一个 index.html 文件在 led-button 文件夹中，webiopi JavaScript 库允许你绑定 GPIO 端口到按键上。你只需要一个单一的<script>标签包括/ webiopi.js。它会自动加载 jQuery（js 库）。

你不需要把 webiopi.js 和 jquery.js 放在 HTML 文件夹下面。你需要的只是<script>标签在您的 index.html 中。webiopi 服务器会自动从提供的 webiopi.js 和 jquery.js 筛选请求 webiopi 资源文件夹中的文件。

该文件是由一些 HTML 标签，包括一个小的 JavaScript 和 CSS 线数。要关心最重要的是匿名的 JavaScript 函数传递给 webiopi JS 库 ready() webiopi()。这使 WebIOPi 和 jQuery 库之前修改 UI 加载。也要把括号当前缀的函数调用与 webiopi()。按钮创建后，我们使用 jQuery 函数将其附加到 HTML 元素中，稍后在<body>标签中声明。

```
1.  <body>
2.      <div id="content" align="center"></div>
3.  </body>
```

一个 ID 为 content 的 div，可使用 jquery 语法 #content 选择该 div。可通过 $("#content").append 加入其他 windows DOM。

1) 
```
// Create a "Light" labeled button for GPIO 17
var button = webiopi().createGPIOButton(17,"Light");
```

创建一个按钮，按钮的 ID 为 hold，名称为 LED1，并注册鼠标按下和鼠标松开事件。createButton 实现部分代码——webiopi.js

```
1.  WebIOPi.prototype.createButton = function (id, label, callback, callbackUp)
    {
2.    var button = $('<button type="button" class="Default">');
3.    button.attr("id", id);
4.    button.text(label);
5.    if (callback != undefined) {
6.      button.bind(BUTTON_DOWN, callback);
7.    }
8.    if (callbackUp != undefined) {
9.      button.bind(BUTTON_UP, callbackUp);
10.   }
11.   return button;
```

```
12. }
```

该代码可创建一个 button 标签——<button type="button" class="Default" id="hold">LED1</button>

2) // Append button to HTML element with ID="controls" using jQuery

$("#controls").append(button);把 button 标签加入 content 中。

创建一个按钮，按钮和 GPIO25 相关联，名称为 LED2。

createGPIOButton 实现部分代码——webiopi.js

```
1.  WebIOPi.prototype.createGPIOButton = function (gpio, label) {
2.   var button = w().createButton("gpio" + gpio, label);
3.   button.bind(BUTTON_DOWN, function(event) {
4.    w().toggleValue(gpio);
5.   });
6.   return button;
7.  }
```

从实现代码中可以看出，GPIOButton 本质为一个 Button，只是这个 Button 的 ID 变为 gpio25。除此之外还给该 button 绑定了一个 mousedown 事件。

toggleValue 实现部分--webiopi.js

```
1.  WebIOPi.prototype.createGPIOButton = function (gpio, label) {
2.   var button = w().createButton("gpio" + gpio, label);
3.   button.bind(BUTTON_DOWN, function(event) {
4.    w().toggleValue(gpio);
5.   });
6.   return button;
7.  }
```

实现部分的代码调用了 digitalWrite。

```
1.  WebIOPi.prototype.digitalWrite = function (gpio, value, callback) {
2.   if (w().GPIO[gpio].func.toUpperCase()=="OUT") {
3.    $.post(w().context + 'GPIO/' + gpio + "/value/" + value, function(data) {
4.     w().updateValue(gpio, data);
5.     if (callback != undefined) {
6.      callback(gpio, data);
7.     }
8.    });
9.   }
10.  else {
11.   //console.log(w().GPIO[gpio].func);
12.  }
```

```
13. }
```

使用 jquery 中的$.post 方法，REST API 为

GPIO/17/value/1　　表示打开 LED

GPIO/17/value/0　　表示关闭 LED

3)

```
// Refresh GPIO buttons

// pass true to refresh repeatedly of false to refresh once

webiopi().refreshGPIO(true);
```

定时刷新界面，如果选择 true。refreshGPIO 实现代码——webiopi.js

```
1.  WebIOPi.prototype.refreshGPIO = function (repeat) {
2.   $.getJSON(w().context + "*", function(data) {
3.    w().updateALT(w().ALT.I2C, data["I2C"]);
4.    w().updateALT(w().ALT.SPI, data["SPI"]);
5.    w().updateALT(w().ALT.UART, data["UART"]);
6.    w().updateALT(w().ALT.ONEWIRE, data["ONEWIRE"]);
7.
8.    $.each(data["GPIO"], function(gpio, data) {
9.        w().updateFunction(gpio, data["function"]);
10.       if ( ((gpio != 4) && ((data["function"] == "IN") || (data["function"]
    == "OUT"))
11.        || ((gpio == 4) && (w().ALT.ONEWIRE["enabled"] == false)))){
12.       w().updateValue(gpio, data["value"]);
13.       }
14.       else if (data["function"] == "PWM") {
15.        w().updateSlider(gpio, "ratio", data["ratio"]);
16.        w().updateSlider(gpio, "angle", data["angle"]);
17.       }
18.   });
19.  });
20.  if (repeat === true) {
21.   setTimeout(function(){w().refreshGPIO(repeat)}, 1000);
22.  }
23. }
```

该部分实现较为复杂，请注意 updateValue 函数——webiopi.js

```
1.  WebIOPi.prototype.updateValue = function (gpio, value) {
2.   w().GPIO[gpio].value = value;
3.   var style = (value == 1) ? "HIGH" : "LOW";
4.   $("#gpio"+gpio).attr("class", style);
```

```
5.  }
```

选择一个 ID 为 gpio??的标签，然后加入一个 class，样式名为 HIGH 或者 LOW。所以和 createButton 创建的 Button 不同，createButton 创建的 Button 标签如下

<button type="button" class="HIGH" id="gpio17">Light</button>

class="HIGH"或 class=“LOW”使用这样的方法可以方便修改按钮的样式，例如按下背景为红色，松开背景为白色。

4) Button 按钮样式

```
button {
            display: block;
            margin: 5px 5px 5px 5px;
            width: 160px;
            height: 45px;
            font-size: 24pt;
            font-weight: bold;
            color: white;
        }
        gpio17.LOW {
            background-color: Black;
        }
        gpio17.HIGH {
            background-color: Blue;
        }
```

## 4.5.4. config 服务器配置

Config 文件需要注意的是：

[SCRIPTS]内容全部注释不许选择指定的 script.py

```
[SCRIPTS]
# Load custom scripts syntax :
# name = sourcefile
#     each sourcefile may have setup, loop and destroy functions and macros
#myscript = /home/pi/webiopi/examples/scripts/macros/script.py
#myscript = /home/pi/WebIOPi-0.7.1/htdocs/app/led-driver/script.py
#------------------------------------------------------------------------------#
```

[REST]中添加 GPIO17 口的配置，这样就能通过 REST 访问和修改端口状态。

```
#------------------------------------------------------------------------------#

[REST]
# By default, REST API allows to GET/POST on all GPIOs
# Use gpio-export to limit GPIO available through REST API
gpio-export = 12, 13, 18, 19, 17
```

## 4.5.5. 测试演示

在 WebIOPi-0.7.1 文件夹中添加一个运行脚本，runWebiopi.sh 如下所示：

```
#!/bin/sh

#进入 webiopi 目录
cd/home/pi/WebIOPi-0.7.1

#将整个网页配置文件夹拷贝至/usr/share/webiopi 目录运行
sudo cp -r -f htdocs /usr/share/webiopi


# 进入 /home/pi/WebIOPi-0.7.1/python 目录
cd/home/pi/WebIOPi-0.7.1/python

# 拷贝 config 配置到运行环境路径
sudo cp -r -f config /etc/webiopi

# 运行 webiopi 后台服务
sudo webiopi -d -c  /etc/webiopi/config
```

完成后，只需要每次在 WebIOPi-0.7.1 目录下运行$./runWebiopi.sh 即可启动服务了。

```
pi@raspberrypi:~ $ cd WebIOPi-0.7.1/
pi@raspberrypi:~/WebIOPi-0.7.1 $ ./runWebiopi.sh
2017-03-22 05:24:53 - WebIOPi - INFO - Starting WebIOPi/0.7.1/Python3.4
2017-03-22 05:24:53 - WebIOPi - DEBUG - Mapping GPIO.digitalCount to REST GET /G
PIO/count
2017-03-22 05:24:53 - WebIOPi - DEBUG - Mapping GPIO.digitalRead to REST GET /GP
IO/%(channel)d/value
2017-03-22 05:24:53 - WebIOPi - DEBUG - Mapping GPIO.digitalWrite to REST POST /
```

图 4-36 运行 webiopi

在同一个路由器下打开网络浏览器，（手机或者 PC 端都可以）访问 192.168.1.117:8000 后输入登录账户和密码（账户：webiopi， 密码：raspberry）即可登录主页。显示内容如下所示：

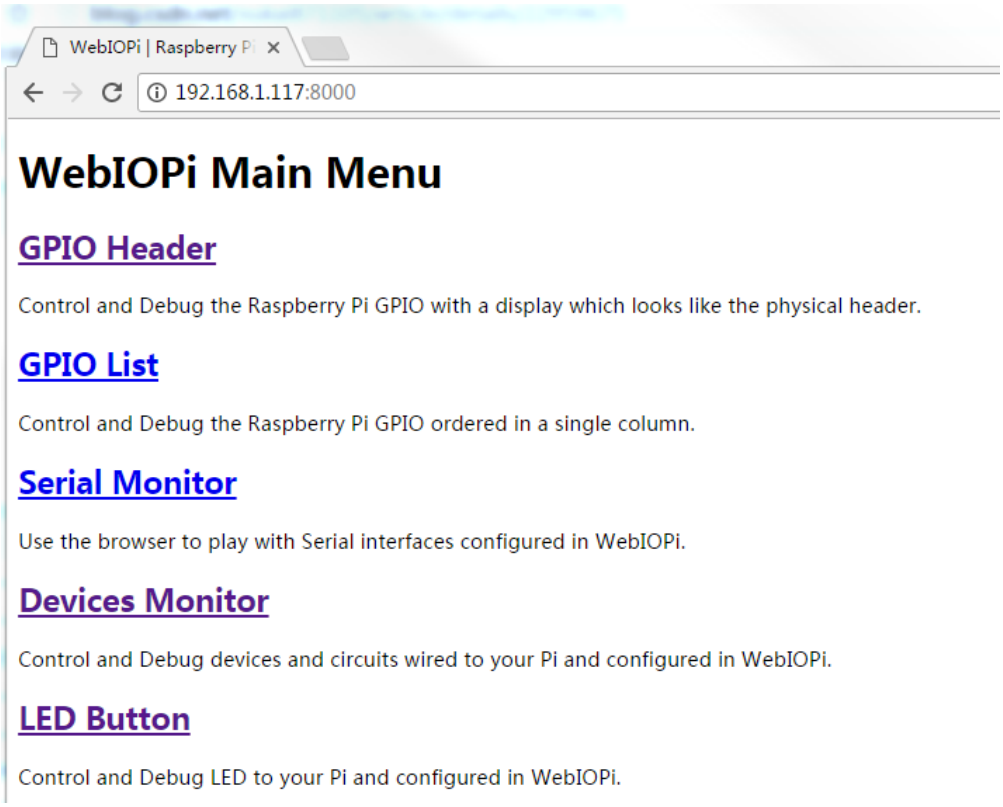



图 4-37 webiopi 主界面

选择点击进入 LED Button 链接：

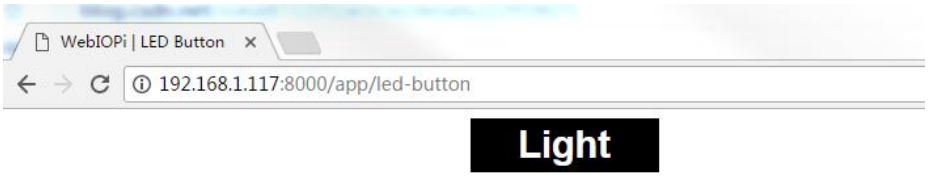



图 4-38 GPIO 按钮控制页面

可以看到一个按钮，这个按钮关联的就是 GPIO17 管脚，确认 GPIO17 管脚硬件上已经连接了 LED 灯后，点击按钮可以切换按钮状态，黑色表示 LED 灯关闭，黄色表示 LED 点亮。

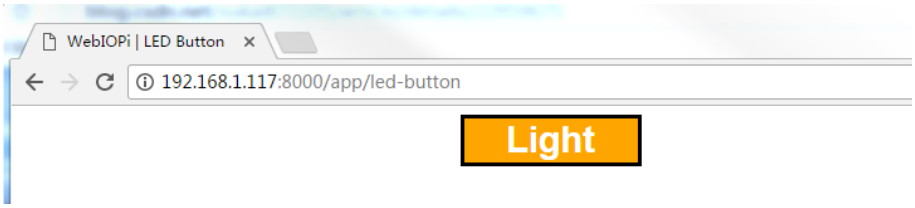



图 4-39 GPIO 按钮控制页面点亮 LED 灯

实际效果和 wiringPi 实验效果类似，但是区别在于 wiringPi 是后台固定时间自动切换 LED 开关，但是 webiopi 却能够使用网页在移动手机或其他任何平台上切换 LED 的状态。也就是从根本上实现了物联网的概念！

图 4-40 webiopi 控制 LED 效果图

## 4.6. 总结

本章节主要引入来 webiopi 这个基于树莓派的物联网开发套件 SDK。它具有强大的快速开发和支持树莓派上应用的特点。也是本书着重介绍的内容，但是对于初学者还是有很多新兴的领域和概念需要慢慢消化吸收。

接下来，结合艾研信息推出的大学生创新实践套件介绍 webiopi 的开发和使用，相信开发者会有写更加具体的感悟。

# 5. 物联网实验套件

艾研信息的物联网套件整合了物联网实验需要涉及到的很多环节，一般至少包含前端传感器电路、传感器的采集数据、系统的电池供电、采集数据上发到网络（或者云服务）、网络平台对数据处理、从网络平台下发到控制前端的反向数据通路、整个系统的移动设备应用呈现（通常是手机端 APP 或者浏览器等方式都可以）。

本章节主要介绍套件中的主要集中物联网模块，他们根据通讯原理可以分为如下几种方式：

1. PWM 控制：例如高亮 LED 灯亮度控制和步进电机控制。
2. I2C 通信：各种传感器诸如光通量、三轴加速度、温湿度等等众多传感器的数据采集；
3. SPI 通信：接近传感器的数据采集或者很多 LCD 液晶模块的通信显示等；

所以我们根据它们的通信方式不同划分成三个类型，进行逐一介绍。同一种类型之间差异不会特别大，只要理解了其中一个模块其他类型的传感器就可以触类旁通了。

## 5.1. PWM 控制

PWM 控制的根本原理就是通过移动端或者网页来实现对于以物联网方式连接到网络上的终端（LED 灯或者步进电机等）实现亮度或者速度的控制功能。

Webiopi 提供了一种 PWM 输出的方式，通过 createRatioSlider 创建一个占空比可控的滑块来实现 PWM 占空比输出。

Webiopi.js 中有如下代码：

```
WebIOPi.prototype.createRatioSlider =function(gpio){
var slider = $('<input type="range" min="0.0" max="1.0" step="0.01">');
slider.attr("id","ratio"+gpio);
slider.bind("change",function(){
    w().pulseRatio(gpio, slider.val());
});
    return slider;
}
```

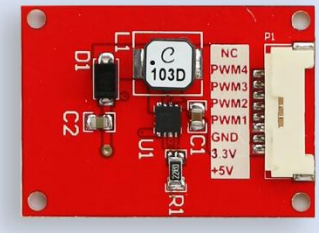
可以在 html 中创建一个滑块来关联一个 GPIO 端口输出一个占空比可调的 PWM 波。

```
WebIOPi.prototype.pulseRatio = function(gpio, ratio, callback) {
    $.post(w().context + 'GPIO/' + gpio + "/pulseRatio/" + ratio, function(data) {
        if (callback != undefined) {
            callback(gpio, data);
        }
    });
}
```

## 5.1.1. 高亮 LED 灯驱动

高亮 LED 驱动模块使用 PWM 接口，模块如下

表 5-1 高亮 LED 驱动模块

| 高亮 LED 驱动模块 | 概况 |
| --- | --- |
|  | 模块使用芯片 TPS61043 驱动 LED<br>使用 PWM 进行控制，可调节 LED 亮度 |

### 5.1.1.1. 实现原理及原理图简介：

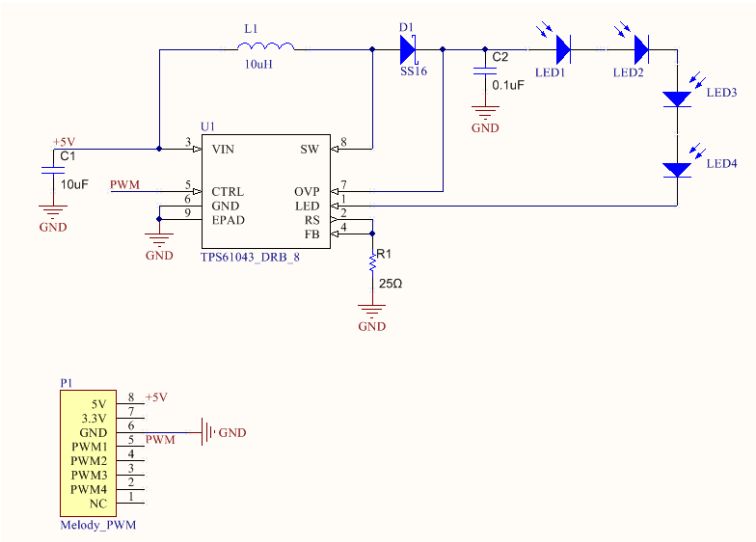



图 5 -1 模块原理图

TPS61043 是 TI 的一款具有恒定电流输出的高频升压转换器，本设计中用于驱动 4 颗 LED 灯，LED 上的电流，通过外部检测电阻 R1 来设置，并由反馈引脚（FB）直接调节，FB 将传感电阻 RS 上的电压调节为 252 mV（典型值）。得

Iout=252mV/R1。

为了控制 LED 亮度，可以通过向控制引脚（CTRL）施加 100Hz 至 50kHz 的频率范围的 PWM（脉宽调制）信号来脉冲化 LED 电流。

#### 5.1.1.2. 实现方法

##### 5.1.1.2.1. 主页添加 LED Driver 链接项

方法和上一章节中 GPIO 实现 IO 控制 LED 开关的准备工作类似。

Htdocs 中存放的是网页访问文件信息，其中/htdocs 文件夹下的 index.html 中可以添加一项 LED Driver 链接项：

<h2><a href="/app/led-driver">LED Driver</a></h2>

Control and Debug LED to your Pi and configured in WebIOP

并且创建一个叫 led-driver 的文件夹，其中创建并存放一个 index.html 和 script.py 文件。

##### 5.1.1.2.2. Index.html 修改

在 WebIOPi-0.7.1\htdocs\app\led-driver\index.html 中可以看到：

```
webiopi().ready(function() {

    // Only for Chrome and Safari, create a slider that pulse out a 0-100% duty cycle ratio on GPIO 13
    var slider = webiopi().createRatioSlider(13);
    $("#controls").append(slider);
```

创建一个 createRatioSlider 端口是树莓派的 GPIO13 口它连接 PWM 输入。并且添加到 controls 中，然后自动刷新。

```
    // Refresh GPIO buttons
    // pass true to refresh repeatedly of false to refresh once
    webiopi().refreshGPIO(true);
```

##### 5.1.1.2.3. Script.py 修改

WebIOPi-0.7.1\htdocs\app\led-driver\script.py 中内容如下所示：

主要目的是将 GPIO13 配置为 PWM 输出。

```
import webiopi
import datetime

# Enable debug output
webiopi.setDebug()

GPIO = webiopi.GPIO

LED_DRIVER_PORT = 13

# setup function is automatically called at WebIOPi startup
def setup():

    GPIO.setFunction(LED_DRIVER_PORT, GPIO.PWM)
    GPIO.pwmWrite(LED_DRIVER_PORT, 0.5)          # set to 50% ratio

# loop function is repeatedly called by WebIOPi
def loop():
    webiopi.sleep(0.1)

# destroy function is called at WebIOPi shutdown
def destroy():
    GPIO.digitalWrite(LED_DRIVER_PORT, GPIO.LOW)
```

图 5-2 Script.py 初始化 PWM

#### 5.1.1.2.4. Config 修改

```
#----------------------------------------------------------------------#
[SCRIPTS]
# Load custom scripts syntax :
# name = sourcefile
#      each sourcefile may have setup, loop and destroy functions and macros
#myscript = /home/pi/webiopi/examples/scripts/macros/script.py
# LED_Driver
myscript = /home/pi/WebIOPi-0.7.1/htdocs/app/led-driver/script.py
```

修改 config 文件中的[SCRIPTS]字段内容，让 webiopi 服务启动的时候自动执行 myscript 路径下的这个 py 脚本文件来实现 GPIO13 口的 PWM 配置。

### 5.1.1.3. 测试演示

配置完成后，将修改过的文件通过 filezilla 提交上传到 raspberry 文件系统中后，直接运行 WebIOPi-0.7.1\runWebiopi.sh 脚本文件。

```
pi@raspberrypi:~/WebIOPi-0.7.1 $ ./runWebiopi.sh
2017-03-22 07:07:55 - WebIOPi - INFO - Starting WebIOPi/0.7.1/Python3.4
2017-03-22 07:07:55 - WebIOPi - DEBUG - Mapping GPIO.digitalCount to REST GET /G
PIO/count
2017-03-22 07:07:55 - WebIOPi - DEBUG - Mapping GPIO.digitalRead to REST GET /GP
IO/%(channel)d/value
```

图 5-3 运行 webiopi

通过网络浏览器访问 192.168.1.117:8000。登陆主页后，点击链接：

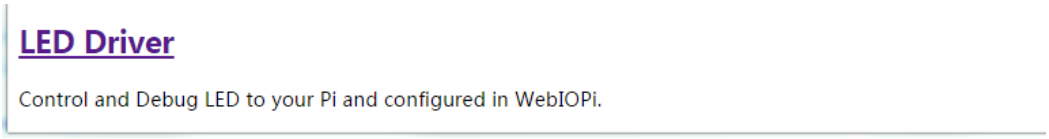



图 5-4 webiopi 主页点击 LED Driver 链接

进入控制页面后看到一个滑块，它能够左右滑动来控制高亮 LED 灯的光强度。

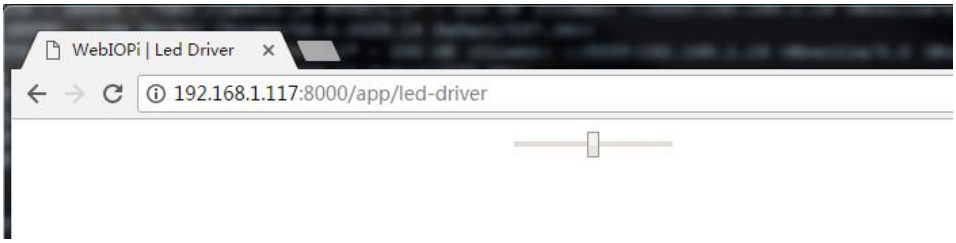



图 5-5 滑块调节亮度

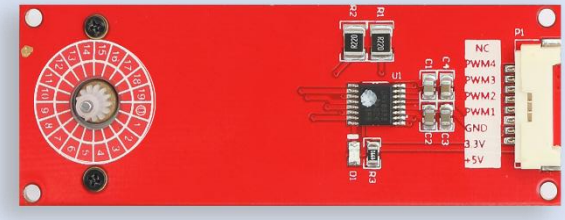
图 5-6 LED 亮度调节效果

## 5.1.2. 步进电机驱动

步进电机模块使用 PWM 接口进行驱动。模块如下

表 5-2 步进电机模块

| 步进电机模块 | 概况 |
| --- | --- |

| | |
|---|---|
|  | 1 模块使用 DRV8833 驱动步进电机<br>2 使用 4 路 PWM 进行驱动，可通过软件改变 PWM 的输出频率和 4 路 PWM 的对应相位，从而改变步进电机的工作速度和方向。 |

### 5.1.2.1. 实现原理及原理图简介

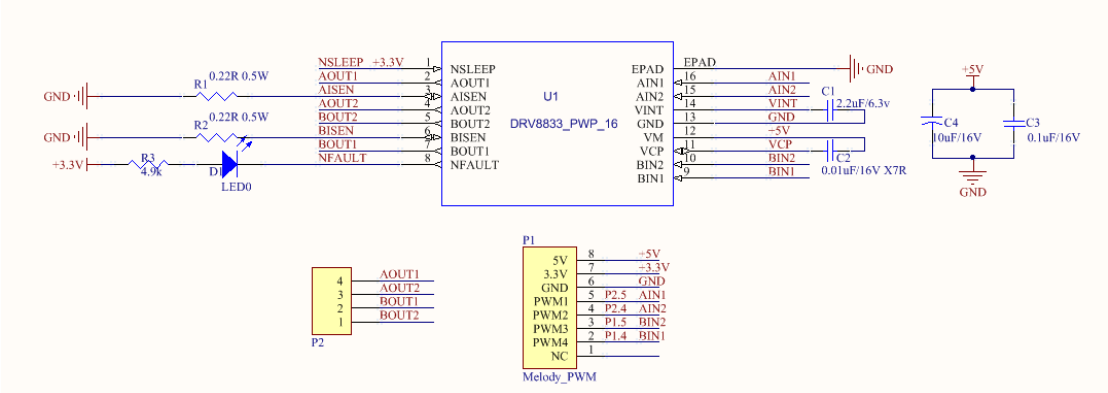



图 5-7 步进电机模块原理图

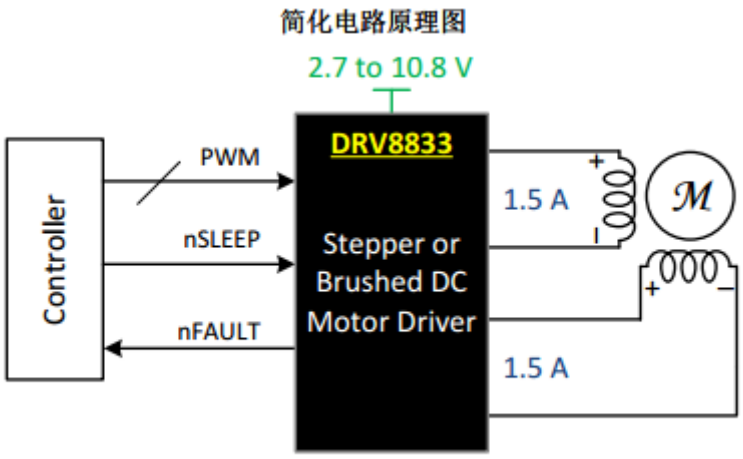



图 5-8 简化原理图

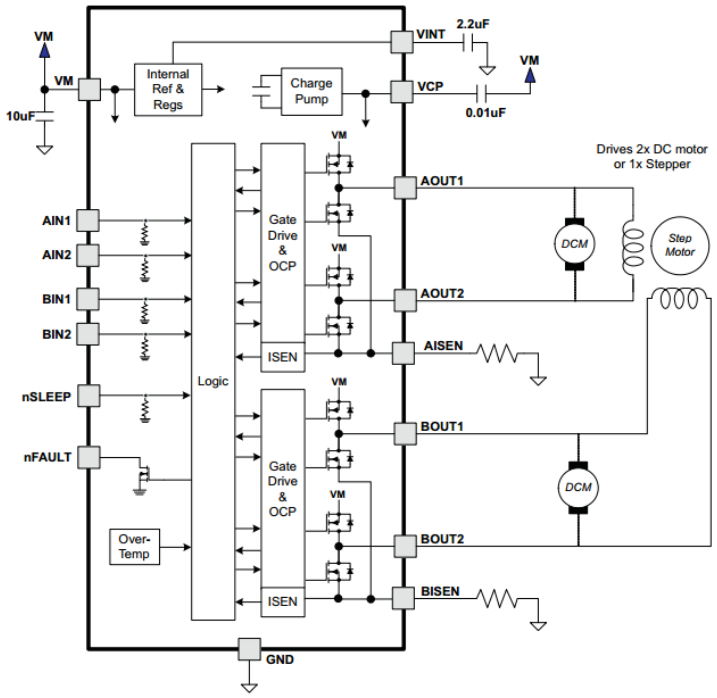



图 5-9 芯片内部原理

本模块采用 TI 的双通道 H 桥电流控制电机驱动器 DRV8833，用于驱动步进电机。每个 H 桥的输出驱动器模块由配置为 H 桥的 N 沟道功率 MOSFET 组成，用于驱动电机绕组。每个 H 桥均具备调节或限制绕组电流的电路。步进电机的步进量以及转速可通过 4 路 PWM 来调节。

根据原理图可知步进电机使用 4 路 PWM 驱动控制，该模块中的 PWM 实现与**错误！未找到引用源。**节**错误！未找到引用源。**中的 PWM 实现不同，需要四组占空比都为 25%但是角度偏移 90° 四组 PWM 配合才能实现。

### 5.1.2.2. 实现方法

在实现步进电机控制的环节，需要应用到 webiopi 宏的概念（Using macros）。具体的方法参考 webiopi 官方网站对应章节。

http://webiopi.trouch.com/Tutorial_Macros.html

当我们能够通过按钮来自动控制开关灯后，我们可能也会遇到需要控制例如开关灯的具体时间或者其他需要配置的内容。这就可以使用 webiopi 的宏来实现。宏其实在 webiopi 中就是一个能够自动约束 REST API 由客户自定义的脚本功能。所以它是可以远程可调用的。它可以远程同时处理 Raspberry Pi 上的复杂运算、手动事件或者多个 GPIO 触发事件，更改设置等等。

如何添加宏：

1) 在 script.py 后面添加：

```
@webiopi.macrodef getLightHours():
 return "%d;%d" % (HOUR_ON, HOUR_OFF)  // 返回含有打开和关闭的两个字段量
@webiopi.macrodef setLightHours(on, off):
 global HOUR_ON, HOUR_OFF
 HOUR_ON = int(on)
 HOUR_OFF = int(off)
 return getLightHours()  //函数转换参数接收和设置的时间和开关点，以及返回的有效时间给反馈到远程用户界面。
```

其中 getLightHours，和 setLightHours 都通过@webiopi.macro 声明。

2) 在 javescript 中调用宏定义

从 JavaScript 中使用宏真的很容易。最困难的部分是与 UI 交互以检索用户输入的数据并显示结果，编辑 index.html 文件。

首先在<body>中添加输入框和文本 Label；

```
<html>
...
<body>
<div align="center">
Turn On at :<input type="text" id="inputOn" /><br/>
Turn Off at: <input type="text" id="inputOff" /><br/>
<div id="controls"></div>
</div>
</body>
</html>
```

图 5-10 index.html 添加输入内容

再次更新 weibiopi.read():

```
webiopi().ready(function() {
    // Following function will process data received from set/getLightHours macro.
    var updateLightHours = function(macro, args, response) {
            var hours = response.split(";");
            // Following lines use jQuery functions
            $("#inputOn").val(hours[0]);
            $("#inputOff").val(hours[1]);
    }
    // Immediately call getLightHours macro to update the UI with current values
    // "getLightHours" refers to macro name
    // [] is an empty array, because getLightHours macro does not take any argument
    // updateLightHours is the callback function, defined above
        webiopi().callMacro("getLightHours", [], updateLightHours);

    // Create a button to call setLightHours macro
    var sendButton = webiopi().createButton("sendButton", "Send", function() {
        // Arguments sent to the macro
            var hours = [$("#inputOn").val(), $("#inputOff").val()];
        // Call the macro
            webiopi().callMacro("setLightHours", hours, updateLightHours);
        });

    // Append the button to the controls box using a jQuery function
        $("#controls").append(sendButton);

    // Append button to HTML element with ID="controls" using jQuery
    $("#controls").append(button);

    // Refresh GPIO buttons
    // pass true to refresh repeatedly of false to refresh once
    webiopi().refreshGPIO(true);
});
```

图 5-11 index.html 创建按键

实现的内容是在网页中添加两个输入框分别绑定 inputOn 和 inputOff 字段。然后创建一个按键来发送两个输入框内的数值给 py 中的两个变量调节灯的开关时间。

根据 webiopi 的这个教程，在本例程中我们需要一个输入量来调节四组 PWM 的频率用于调节步进电机的运行速度。就可以去第一个字段为标准。

#### 5.1.2.2.1. 主页添加 Step Driver 链接

方法同上这里不再赘述。

#### 5.1.2.2.2. Index.html 修改

添加宏定义和数字输入框用于获取步进电机的运行频率。Index 内容已经在上面讲解了，这里略过。

#### 5.1.2.2.3. Srcipt.py 修改

Py 中我们可以获取到来由 index 传回来的控制电机运行速度的字段。接下来就需要实现四组同占空比不同角度的 PWM 波。

首先根据硬件电路配置了四组 PWM 输出端口：

```
MOTOR_DRIVER_1 = 13 # STEP Motor Diver Line GPIO13 PWM1
MOTOR_DRIVER_2 = 18 # GPIO19 PWM3
MOTOR_DRIVER_3 = 19 # GPIO19 PWM2
MOTOR_DRIVER_4 = 12 # GPIO12 PWM4
```

图 5-12 定义电机端口

其次在 def setup 段中初始化四个端口：

```
#
GPIO.setFunction(MOTOR_DRIVER_1, GPIO.OUT)
GPIO.setFunction(MOTOR_DRIVER_2, GPIO.OUT)
GPIO.setFunction(MOTOR_DRIVER_3, GPIO.OUT)
GPIO.setFunction(MOTOR_DRIVER_4, GPIO.OUT)
```

图 5-13 初始化端口

再次利用 def loop 段的循环机制，创建一个变量不断切换四个端口的高低电平，保证任何状态只有一个端口出于高电平实现四组同占空比不同角度输出 PWM。然后将获取到的 HOUR_ON 字段作为循环的时间标准。假如 HOUR_ON=1

```
# 创建一个变量不断切换四个端口的高低电平，保证任何状态只有
# 一个端口出于高电平实现四组同占空比不同角度输出PWM
global tick
if tick == 3:
    tick = 0
else:
    tick += 1

print("tick = ", tick)
print("LOOP HOUR_ON = ",HOUR_ON)

if tick == 0:
    GPIO.digitalWrite(MOTOR_DRIVER_1, GPIO.HIGH)
    GPIO.digitalWrite(MOTOR_DRIVER_2, GPIO.LOW)
    GPIO.digitalWrite(MOTOR_DRIVER_3, GPIO.LOW)
    GPIO.digitalWrite(MOTOR_DRIVER_4, GPIO.LOW)
elif tick == 1:
    GPIO.digitalWrite(MOTOR_DRIVER_1, GPIO.LOW)
    GPIO.digitalWrite(MOTOR_DRIVER_2, GPIO.HIGH)
    GPIO.digitalWrite(MOTOR_DRIVER_3, GPIO.LOW)
    GPIO.digitalWrite(MOTOR_DRIVER_4, GPIO.LOW)
elif tick == 2:
    GPIO.digitalWrite(MOTOR_DRIVER_1, GPIO.LOW)
    GPIO.digitalWrite(MOTOR_DRIVER_2, GPIO.LOW)
    GPIO.digitalWrite(MOTOR_DRIVER_3, GPIO.HIGH)
    GPIO.digitalWrite(MOTOR_DRIVER_4, GPIO.LOW)
else:
    GPIO.digitalWrite(MOTOR_DRIVER_1, GPIO.LOW)
    GPIO.digitalWrite(MOTOR_DRIVER_2, GPIO.LOW)
    GPIO.digitalWrite(MOTOR_DRIVER_3, GPIO.LOW)
    GPIO.digitalWrite(MOTOR_DRIVER_4, GPIO.HIGH)

# gives CPU some time before looping again
webiopi.sleep(0.1 * HOUR_ON)
```

图 5-14 电机四组 PWM 运行逻辑

### 5.1.2.2.4. Config 修改

```
[SCRIPTS]

# Load custom scripts syntax :
```

```
# STEP Motor
myscript = /home/pi/WebIOPi-0.7.1/htdocs/app/step-motor/script.py
```

如上所示，将[SCRIPTS]字段的 myscript 路径修改为上面修改完成的 script.py 这样启动 webiopi 后台服务会自动启动 script.py 脚本程序。

#### 5.1.2.3. 测试演示

配置完成后，将修改过的文件通过 filezilla 提交上传到 raspberry 文件系统中后，直接运行 WebIOPi-0.7.1\runWebiopi.sh 脚本文件。

```
pi@raspberrypi:~/WebIOPi-0.7.1 $ ./runWebiopi.sh
2017-03-22 07:07:55 - WebIOPi - INFO - Starting WebIOPi/0.7.1/Python3.4
2017-03-22 07:07:55 - WebIOPi - DEBUG - Mapping GPIO.digitalCount to REST GET /G
PIO/count
2017-03-22 07:07:55 - WebIOPi - DEBUG - Mapping GPIO.digitalRead to REST GET /GP
IO/%(channel)d/value
```

图 5-15 运行 webiopi

通过网络浏览器访问 192.168.1.117:8000。登陆主页后，点击链接：

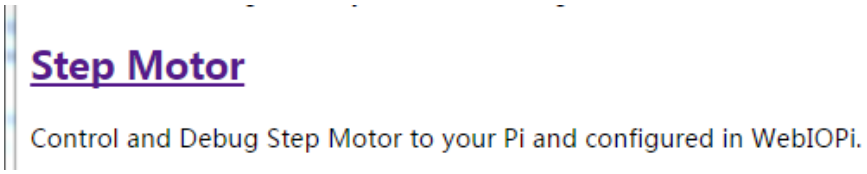



图 5-16 访问电机链接

进入控制页面后看到两个输入框，其中第一个 Turn On at 后的输入框的内容就是步进电机的每步的时间间隔 0.1S*8 = 0.8 秒。可以自行修改里面的数字并且点击 Send 按钮就能够调整步进电机的运行速度了。

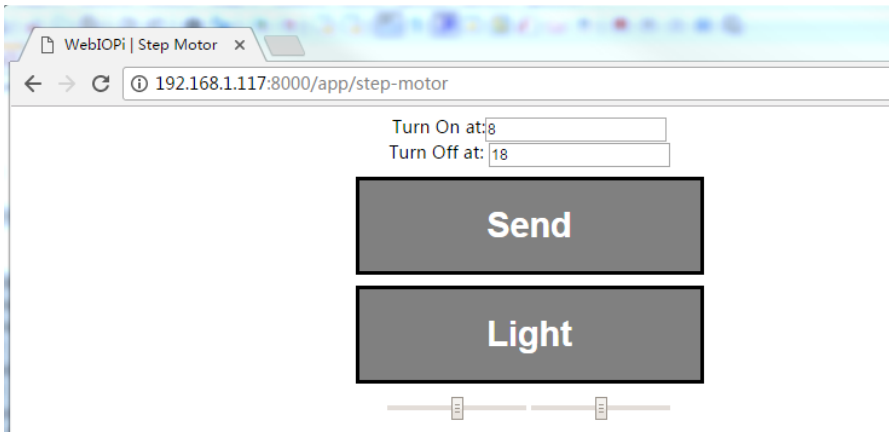



图 5-17 步进电机控制页面

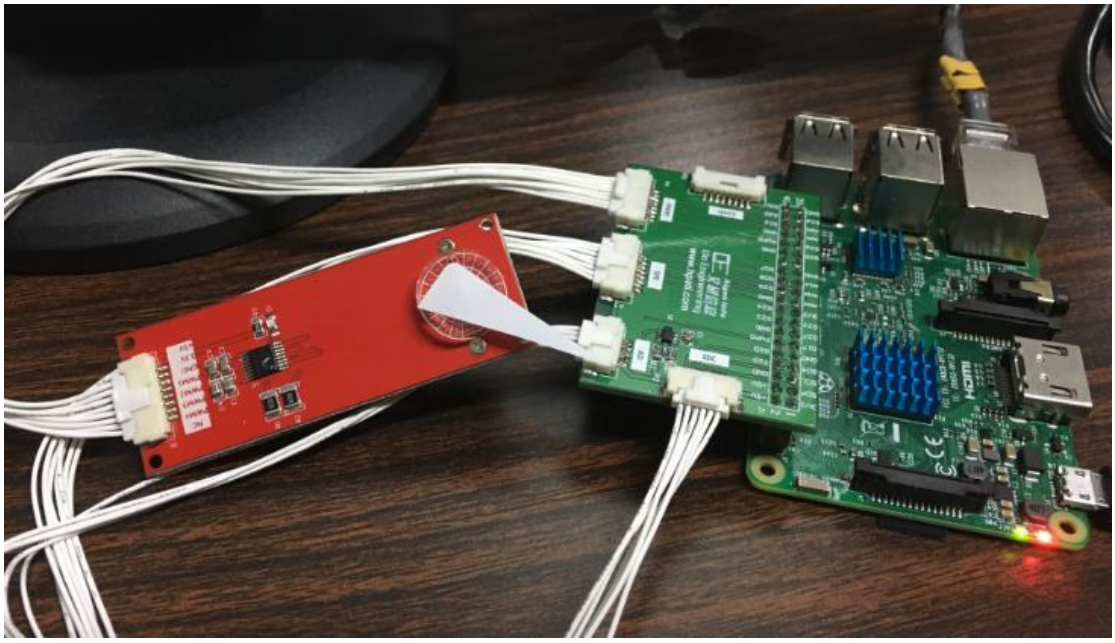


图 5-18 步进电机控制效果

实验视频：

## 5.2. I2C 通信

I2C 总线（I2C bus，Inter-IC bus）是一个双向的两线连续总线，提供集成电路（ICs）之间的通信线路。I2C 总线是一种串行扩展技术，广泛应用于各种传感器中。I2C 总线的意思是“完成集成电路或功能单元之间信息交换的规范或协议”。Philips 公司推出的 I2C 总线采用一条数据线（SDA），加一条时钟线（SCL）来完成数据的传输及外围器件的扩展；对各个节点的寻址是软寻址方式，节省了片选线，标准的寻址字节 SLAM 为 7 位，可以寻址 127 个单元。

### 5.2.1. 如何创建 I2C 传感器驱动

如果希望创建属于自己的 I2C 传感器驱动，首 Webiopi 自带来例如 tmpxxx.py 的 I2C 驱动脚本，我们需要做的就是模仿该方式创建自己传感器的脚本驱动。

先要理解 webiopi 设备的管理运行模式。这里可以参看 webiopi 官网的教程关于 Using devices 一章节

http://webiopi.trouch.com/Tutorial_Devices.html

简单来讲分为如下几部：

- 在/etc/webiopi/config 中创建并添加传感器

[DEVICES]

tmp = TMP102

理解为创建一个 TMP102 类型的设备，并命名为 tmp；当设备被命名为在配置文件中,驱动程序会自动叫初始化设备和映射到 REST API。我们需要使用 webiopi.deviceInstance(名字)函数在 Python 脚本检索给定名称的设备对象实例。然后我们可以使用的功能驱动程序,不同的接口支持。接下来我们就以 TMP102 为例。看它出现在哪些配置中。

#### 5.2.1.1. 编写驱动脚本

在 webIOPi-0.7.1\python\webiopi\devices\sensor 文件中，我们发现有个一叫做 tmpXXX.py 的脚本驱动。


它便是温度传感器的最原始驱动定义。打开该文件。

```
from webiopi.devices.i2c import I2C
from webiopi.devices.sensor import Temperature
from webiopi.devices.sensor import Humidity

class TMP102(I2C, Temperature):
    def __init__(self, slave=0x48):
        I2C.__init__(self, toint(slave))

    def __str__(self):
        return "TMP102(slave=0x%02X)" % self.slave

    def __getKelvin__(self):
        return self.Celsius2Kelvin()

    def __getCelsius__(self):
        d = self.readBytes(2)
        count = ((d[0] << 4) | (d[1] >> 4)) & 0xFFF
        return signInteger(count, 12)*0.0625

    def __getFahrenheit__(self):
        return self.Celsius2Fahrenheit()
```

图 5-19 tmpXXX.py 驱动脚本

首先它继承来 I2C 协议和一个叫 Temperature 的定义，并且在类中实现了它们的一些方法。比如初始化 I2C 地址为 0x48.而关于 Temperature 的定义出现在了 WebIOPi-0.7.1\python\webiopi\devices\sensor__init__.py 中：

```
class Temperature():
    def __family__(self):
        return "Temperature"

    def __getKelvin__(self):
        raise NotImplementedError

    def __getCelsius__(self):
        raise NotImplementedError

    def __getFahrenheit__(self):
        raise NotImplementedError


    @request("GET", "sensor/temperature/k")
    @response("%.02f")
    def getKelvin(self):
        return self.__getKelvin__()

    @request("GET", "sensor/temperature/c")
    @response("%X")
    def getCelsius(self):
        return self.__getCelsius__()

    @request("GET", "sensor/temperature/f")
    @response("%.02f")
    def getFahrenheit(self):
        return self.__getFahrenheit__()
```

图 5-20 Temperature 类定义

#### 5.2.1.2. Webiopi 库添加驱动脚本

在 WebIOPi-0.7.1\python\WebIOPi.egg-info\SOURCE.TXT 中能够看到 webiopi/devices/sensor/tmpXXX.py 的定义，在我的理解应该是 webIopi 在安装时它会根据 SOURCE.TXT 的路径定义来查找，拷贝，编译这些源文件到系统环境中去。所以如果要创建属于自己的传感器驱动，这里添加对于的脚本驱动是必不可少的。

#### 5.2.1.3. 传感器管理中初始化驱动脚本

在 WebIOPi-0.7.1\python\webiopi\devices\sensor__init__.py 中不仅仅有每种传感器的 class 定义，在最后还有所有 Devices 的定义，所有的在 config 中的定义都必须来自于这里不然系统无法识别设备。

DRIVERS["tmpXXX"] = ["TMP75", "TMP102", "TMP275", "HDC1080"]

标明关于"TMP75", "TMP102", "TMP275", "HDC1080"这些设备都来自于 tmpXXX 的脚本驱动。

#### 5.2.1.4. Webiopi.js 中添加新设备

在 WebIOPi-0.7.1\htdocs\app\devices-monitor\index.html 的内容如下：

```
<script type="text/javascript">
webiopi().ready(function() {
    $.get("/devices/*", function(data) {//获取所有devices下面挂载的设备
        var content = $("#content");
        for (i in data) {//遍历所有挂载的设备并将不属于"Serial"的类型全部append到content内
            if (data[i].type != "Serial") {
                var device = webiopi().newDevice(data[i].type, data[i].name);
                if (device) {
                    device.element = $("<div>");
                    content.append(device.element);
                    device.refreshUI();
                }
            }
        }
    });
});
</script>
```

图 5-21 index.html 初始化设备管理

只要是 config 中配置了的所有非 serial 设备我们都会显示出来。在对应的 WebIOPi-0.7.1\htdocs\webiopi.js 中我们也需要有响应的准备如下：

```
function Temperature(name) {
    this.name = name;
    this.url = "/devices/" + name + "/sensor";
    this.refreshTime = 5000;
}

Temperature.prototype.toString = function() {
    return this.name + ": Temperature";
}

Temperature.prototype.getKelvin = function(callback) {
    $.get(this.url + "/temperature/k", function(data) {
        callback(this.name, data);
    });
}

Temperature.prototype.getCelsius = function(callback) {
    $.get(this.url + "/temperature/c", function(data) {
        callback(this.name, data);
    });
}

Temperature.prototype.getFahrenheit = function(callback) {
    $.get(this.url + "/temperature/f", function(data) {
        callback(this.name, data);
    });
}

Temperature.prototype.refreshUI = function() {
    var temp = this;
```

图 5-22 webiopi.js 定义设备

并且在 newDevices 的枚举类型时一定也要添加相应类型的类型选项：

```
WebIOPi.prototype.newDevice = function(type, name) {
    if (type == "ADC") {
        return new ADC(name);
    }

    if (type == "DAC") {
        return new DAC(name);
    }

    if (type == "PWM") {
        return new PWM(name);
    }

    if (type == "GPIOPort") {
        return new GPIOPort(name);
    }

    if (type == "Temperature") {
        return new Temperature(name);
    }

    /*TODO ZK ADD Acceleration*/
    if (type == "Acceleration") {
        return new Acceleration(name);
    }
```

图 5-22 newDevices 枚举设备类型

#### 5.2.1.5. Config 中创建设备

如果以上准备都已配置结束了，最后一部就是在 config 添加一个传感器了。

#----------------------------------------------------------------------#

[DEVICES]

#温度传感器

temp = TMP102

#### 5.2.1.6. 运行分析

如果是添加了新的传感器到原有的 webiopi 系统中，那么需要重新安装一次 webiopi 不然是不能生效的，安装方法自行从上面章节查找。假设重新安装成功后，运行./runWebiopi.sh 在打印信息中我们能够看到如下 Debug 内容：

```
2017-03-23 06:06:48 - WebIOPi - DEBUG - Mapping TMP102(slave=0x48).getCelsius to REST GET /devices/temp/sens
or/temperature/c
2017-03-23 06:06:48 - WebIOPi - DEBUG - Mapping TMP102(slave=0x48).getFahrenheit to REST GET /devices/temp/s
ensor/temperature/f
2017-03-23 06:06:48 - WebIOPi - DEBUG - Mapping TMP102(slave=0x48).getKelvin to REST GET /devices/temp/senso
r/temperature/k
2017-03-23 06:06:48 - WebIOPi - INFO - Temperature - TMP102(slave=0x48) mapped to REST API /devices/temp
```

图 5-23 运行 webiopi

意义就在于创建了 Restful，可以供 web 端不断的 get 到温度传感器的数据。打开网页点击 Devices Monitor：

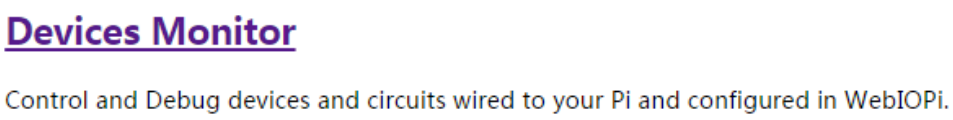


图 5-24 进入设备管理链接

可以看到如下的显示内容：


图 5-25 显示温度传感器数据

这样我们就能够通过移动端获取到挂载在树莓派上的任何传感器的数据了。

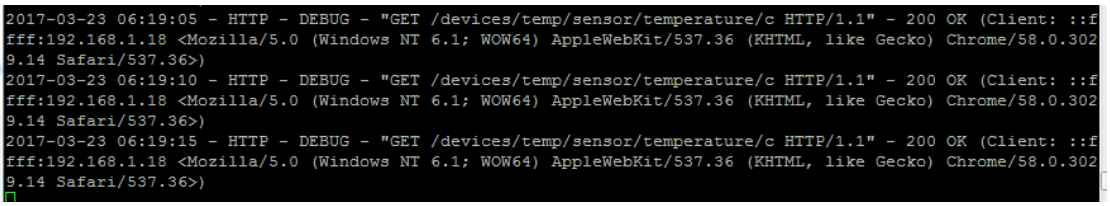


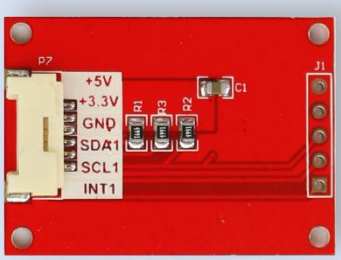

图 5-26 后台打印温度传感器数据

查看后台打印信息我们能够看到，假如我们停留在 Devices Monitor 页面，后台会间隔 5 秒不断地更新温度传感器的数据。假如退出来网页，则更新停止。

## 5.2.2. 光感传感器（OPT3001）

光感模块使用 I2C 接口与树莓派进行通信，可读取模块的光照强度数据。模块介绍如下：

表 5-4 光感模块

| 光感模块 | 概况 |
| --- | --- |
|  | 1 模块使用芯片 OPT3001<br>2 使用 I2C 接口通信<br>3 测量范围：0.01lux 至 83k lux<br>4 23 位有效动态范围，具有自动增益范围设置功能 |

### 5.2.2.1. 实现原理及原理图简介

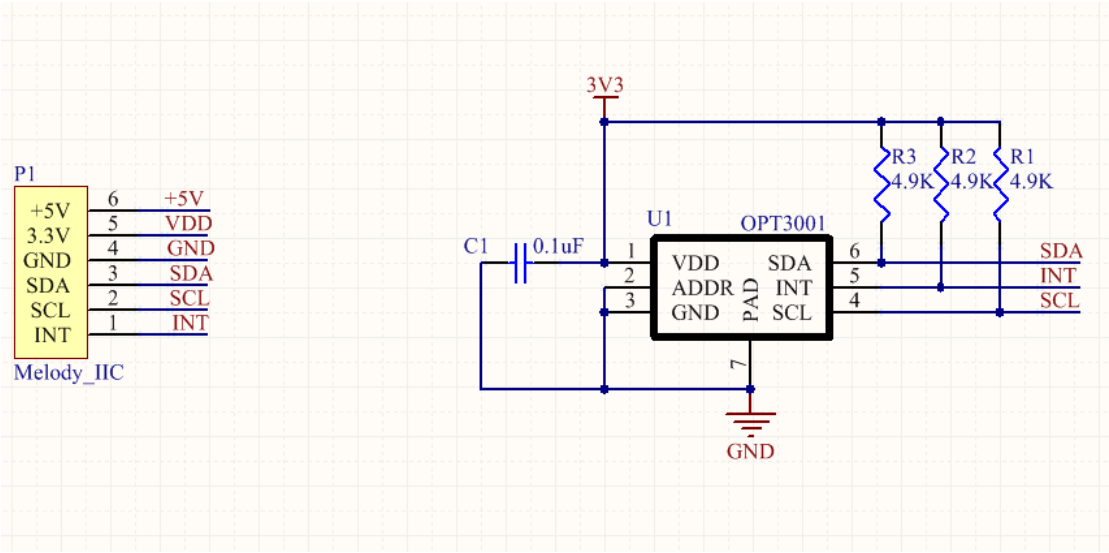



图 5-27 光感模块原理图

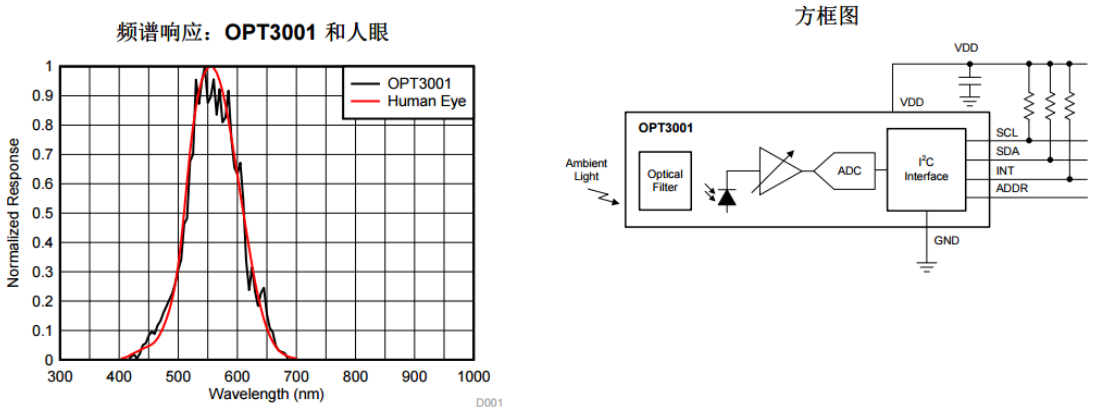



图 5-28 频谱相应和简化原理图

OPT3001 传感器用于测量可见光的密度。OPT3001 器件兼具精密的频谱响应和较强的 IR 阻隔功能，，因此能够如人眼般准确测量光强且不受光源影响。OPT3001 专门针对构建基于光线的人眼般体验的系统而设计，是人眼匹配度低且红外阻隔能力差的光电二极管、光敏电阻或其它环境光传感器的首选理想替代产品。

测量范围可达 0.01lux 至 83k lux，且内置有满量程设置功能，无需手动选择满量程范围。数字输出通过 I2C 接口于 Raspberry 连接即可。

### 5.2.2.2. 实现方法

上一小节中我们介绍了原有的温度传感器驱动的工作流程。再创建一个新的传感器就会相对容易点，只需要按照原有的方式依葫芦画瓢即可。

- 编写驱动脚本：

```
#定义 OPT3001 光感传感器
classOPT3001(I2C, Luminosity):
    def__init__(self, slave=0x44):#I2C 初始化传感器，I2C 地址为 0X44)
        I2C.__init__(self, toint(slave))
        config =[0xc4,0x10]        #芯片初始化参数
        self.writeRegisters(0x01,config)#将初始化参数写入寄存器 0x01
#本传感器描述字符串
    def__str__(self):
        return"OPT3001(slave=0x%02X)"% self.slave
#获取流明原始数据
    def__getLux__(self):
        d0 = self.readRegisters(0x00,2)
        count =((d0[0]<<4)|(d0[1]>>4))&0xFFFF
            return count
```

- Webiopi 库添加驱动脚本

  在 WebIOPi-0.7.1\python\WebIOPi.egg-info\SOURCE.TXT 中添加：

  webiopi/devices/sensor/optXXX.py 一项

- 传感器管理中初始化驱动脚本

  添加新的传感器：DRIVERS["optXXX"] = ["OPT3001"]

  添加 Luminosity 类定义：

```
classLuminosity():
def__family__(self):
return"Luminosity"

def__getLux__(self):
raise NotImplementedError

@request("GET","sensor/luminosity/lux")
@response("%.02f")
defgetLux(self):
    return self.__getLux__()
```

- Webiopi.js 中添加新设备，newDevice 中添加新的类型定义

```
if(type =="Luminosity"){
    returnnew Luminosity(name);
```

```
    }
function Luminosity(name){
    this.name = name;
    this.url ="/devices/"+ name +"/sensor";
    this.refreshTime =1000;
}
Luminosity.prototype.toString=function(){
    returnthis.name +": Luminosity";
}
Luminosity.prototype.getLux =function(callback){
    $.get(this.url +"/luminosity/lux",function(data){
        callback(this.name, data);
    });
}
Luminosity.prototype.refreshUI =function(){
    var lum =this;
    varelement=this.element;

    if((element!=undefined)&&(element.header ==undefined)){
        element.header = $("<h3>"+this+"</h3>");
        element.append(element.header);
    }

    this.getLux(function(name, data){
        if(element!=undefined){
            element.header.text(lum +": "+ data +"lux");
        }
        setTimeout(function(){lum.refreshUI()}, lum.refreshTime);
    });
    }
```

- config 中创建设备：

  #光感传感器

  opt = OPT3001

### 5.2.2.3. 5.2.1.6. 测试演示

重新安装一次 webiopi，安装方法自行从上面章节查找。假设重新安装成功后，运行./runWebiopi.sh 在打印信息中我们能够看到如下 Debug 内容：

```
2017-03-23 06:51:16 - WebIOPi - DEBUG - Setup GPIO 23
2017-03-23 06:51:16 - WebIOPi - DEBUG - Setup GPIO 24
2017-03-23 06:51:16 - WebIOPi - DEBUG - Setup GPIO 25
2017-03-23 06:51:16 - WebIOPi - DEBUG - Mapping OPT3001(slave=0x44).getLux to REST GET /devices/opt/sensor/l
uminosity/lux
2017-03-23 06:51:16 - WebIOPi - INFO - Luminosity - OPT3001(slave=0x44) mapped to REST API /devices/opt
```

图 5-29 配置了 OPT3001 到 REST 上

意义就在于创建了 Restful，可以供 web 端不断的 get 到温度传感器的数据。打开网页点击 Devices Monitor：

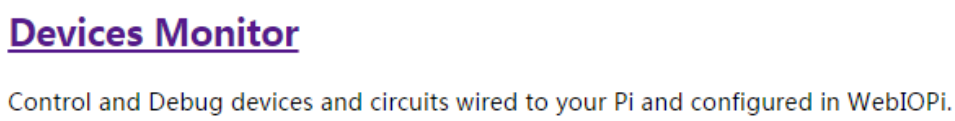



图 5-30 点击设备管理链接

可以看到如下的显示内容：

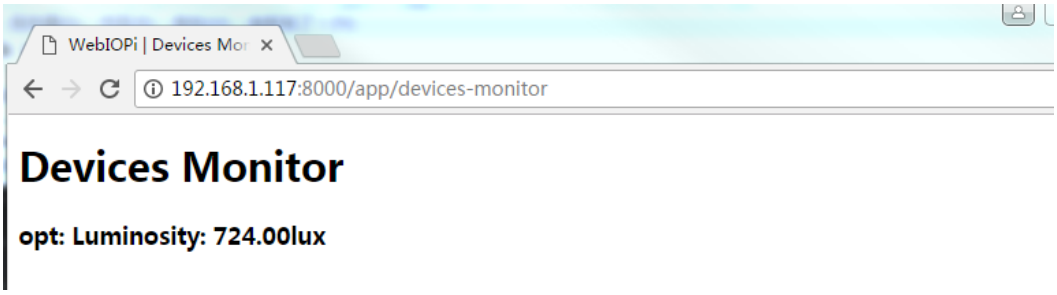



图 5-31 显示光感强度

手动调节光感的方向，会发现有数值的不断变化。这样我们就能够通过移动端获取到挂载在树莓派上的任何传感器的数据了。

2017-03-23 06:51:34 - HTTP - DEBUG - "GET /devices/opt/sensor/luminosity/lux HTTP/1.1" - 200 OK (Client: ::ffff:192.168.1.18 <Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.14 Safari/537.36>)

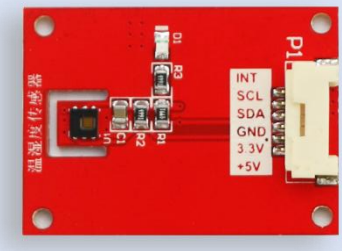
查看后台打印信息我们能够看到，假如我们停留在 Devices Monitor 页面，后台会间隔 5 秒不断地更新光感传感器的数据。假如退出来网页，则更新停止。

### 5.2.3. 温湿度传感器（HDC1080）

温湿度传感器模块使用 I2C 接口与 Raspberry 进行通信，可读取模块的温度和湿度数据。模块如下：

表 5-5 温湿度传感器模块

| 温湿度传感器模块 | 概况 |
| --- | --- |




1 模块使用芯片 HDC1080

2 使用 I2C 接口，读取模块的温湿度数据

3 相对湿度精度达到±2%

4 温度精度达到±0.2℃

5 14 位测量分辨率

### 5.2.3.1. 实现原理及原理图简介

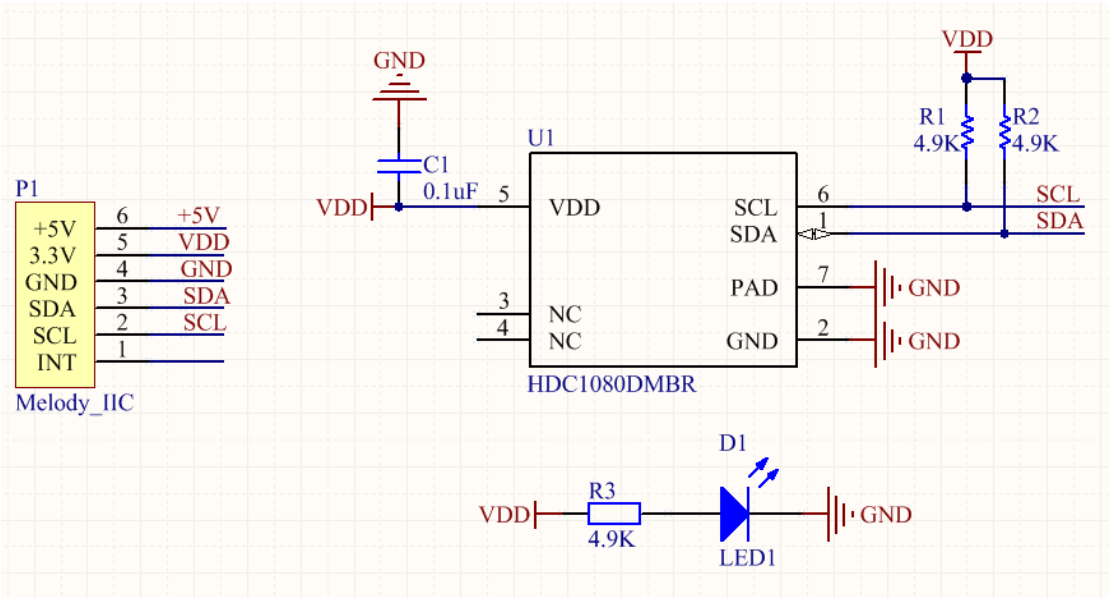



图 5-22 温湿度传感器模块原理图

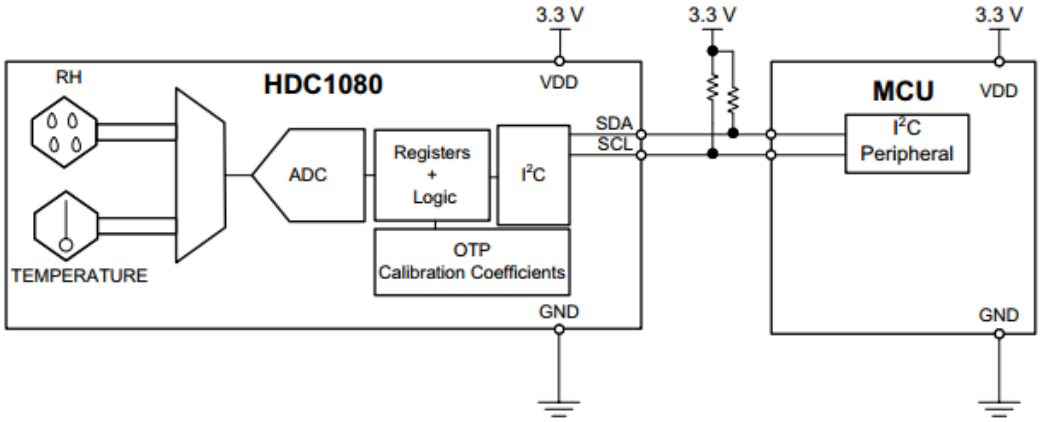



图 5-33 原理简化图

HDC1080 集成了温度传感器和数字湿度传感器，并能够以超低功耗提供出色的测量精度，相对湿度精度为 ±2%（典型值），温度精度为 ±0.2° C（典型值）。

具有 14 位测量分辨率。通过 I2C 接口，与 Raspberry 连接即可，使用便利。

#### 5.2.3.2. 实现方法

➢ 编写驱动脚本：

温湿度传感器基本和温度传感器类似，所以在这里我们不在单独创建一个全新的 py 脚本而是在 tmpXXX.py 中直接添加一个新的 class 类，继承至 I2C, Temperature, Humidity 三个类。

```
#添加传感器模块，继承温度和湿度
classHDC1080(I2C, Temperature, Humidity):
def__init__(self, slave=0x40):
        I2C.__init__(self, toint(slave))
def__str__(self):
return"HDC1080(slave=0x%02X)"% self.slave
def__getCelsius__(self):
#self.writeByte(0x00)
        d = self.readBytes(2)
        count =((d[0]<<4)|(d[1]>>4))&0xFFFF
return count
def__getHumidity__(self):
# self.writeByte(0x01)
        d0 = self.readBytes(2)
        count =((d0[0]<<4)|(d0[1]>>4))&0xFFFF
return count
def__getHumidityPercent__(self):
        d0 = self.readRegisters(0x32,2)
        count =((d0[0]<<4)|(d0[1]>>4))&0xFFFF
return count
```

➢ Webiopi 库添加驱动脚本

在 WebIOPi-0.7.1\python\WebIOPi.egg-info\SOURCE.TXT 中添加：

webiopi/devices/sensor/tmpXXX.py 一项

➢ 传感器管理中初始化驱动脚本：

设备管理项内添加 HDC1080；

DRIVERS["tmpXXX"] = ["TMP75", "TMP102", "TMP275", "HDC1080"]

```
classHumidity():
def__family__(self):
return"Humidity"
def__getHumidity__(self):
raise NotImplementedError
```

```
@request("GET","sensor/humidity/float")
@response("%f")
defgetHumidity(self):
return self.__getHumidity__()
@request("GET","sensor/humidity/percent")
@response("%d")
defgetHumidityPercent(self):
return self.__getHumidity__()*100
```

- Webiopi.js 中添加新设备：

  Temperature 已经添加过了，只需要再次添加 Luminosity 定义。

```
function Luminosity(name){
    this.name = name;
    this.url ="/devices/"+ name +"/sensor";
    this.refreshTime =1000;
}
Luminosity.prototype.toString=function(){
    returnthis.name +": Luminosity";
}
Luminosity.prototype.getLux =function(callback){
    $.get(this.url +"/luminosity/lux",function(data){
        callback(this.name, data);
    });
}
Luminosity.prototype.refreshUI =function(){
    var lum =this;
    varelement=this.element;

    if((element!=undefined)&&(element.header ==undefined)){
        element.header = $("<h3>"+this+"</h3>");
        element.append(element.header);
    }

    this.getLux(function(name, data){
        if(element!=undefined){
            element.header.text(lum +": "+ data +"lux");
        }
        setTimeout(function(){lum.refreshUI()}, lum.refreshTime);
    });
}
```

- Config 中创建设备：

#温湿度传感器

hdc = HDC1080

#### 5.2.3.3. 测试演示

将编辑的各种文档上传到 Raspberry 上对应的路径下重新编译 webiopi。完成后运行./runWebiopi.sh 脚本。在打印信息中我们能够看到如下 Debug 内容：

```
2017-03-23 07:23:50 - WebIOPi - DEBUG - Mapping HDC1080(slave=0x40).getCelsius to REST GET /devices/hdc/sens
or/temperature/c
2017-03-23 07:23:50 - WebIOPi - DEBUG - Mapping HDC1080(slave=0x40).getFahrenheit to REST GET /devices/hdc/s
ensor/temperature/f
2017-03-23 07:23:50 - WebIOPi - DEBUG - Mapping HDC1080(slave=0x40).getHumidity to REST GET /devices/hdc/sen
sor/humidity/float
2017-03-23 07:23:50 - WebIOPi - DEBUG - Mapping HDC1080(slave=0x40).getHumidityPercent to REST GET /devices/
hdc/sensor/humidity/percent
2017-03-23 07:23:50 - WebIOPi - DEBUG - Mapping HDC1080(slave=0x40).getKelvin to REST GET /devices/hdc/senso
r/temperature/k
```

图 5-44 启动 webiopi

意义就在于创建了 Restful，可以供 web 端不断的 get 到温湿度传感器的数据。打开网页点击 Devices Monitor：

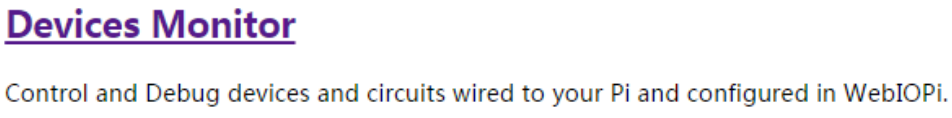



图 5-55 进入设备管理链接

可以发现显示的数据出现异常。然后我们使用 i2cdetect 查看挂载的很多次 080 传感器的数据：

```
pi@raspberrypi:~/WebIOPi-0.7.1 $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

HDC1080 的端口号是 0x40，使用 dump 打印 I2C 总线地址的所有数据发现：

```
pi@raspberrypi:~/WebIOPi-0.7.1 $ i2cdump -y 1 0x40 w
     0,8  1,9  2,a  3,b  4,c  5,d  6,e  7,f
00: XXXX XXXX 0010 XXXX XXXX XXXX XXXX XXXX
08: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
10: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
18: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
20: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
28: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
30: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
38: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
40: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
48: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
50: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
58: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
60: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
68: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
70: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
78: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
80: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
88: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
90: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
98: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
a0: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
a8: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
b0: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
b8: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
c0: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
c8: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
d0: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
d8: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
e0: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
e8: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
f0: XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
f8: XXXX XXXX XXXX c900 9e02 807d 4954 5010
```

除了寄存器地址 0x02 上的 config 数据和最后设备描述数据能够成功的读取，而寄存器 0x01，0x02 上面的温湿度数据没有办法读取。

```
pi@raspberrypi:~/WebIOPi-0.7.1 $ i2cget -y 1 0x40 0x01
Error: Read failed
pi@raspberrypi:~/WebIOPi-0.7.1 $ i2cget -y 1 0x40 0x00
Error: Read failed
```

找了很多论坛内的答疑貌似 HDC1080 暂时还不能很好的解决 Raspberry 3 的支持问题，用 i2cdetect 工具也不能够连续有效的获取到温湿度数据内容。参看：

https://e2e.ti.com/support/arm/sitara_arm/f/791/p/490715/1771752

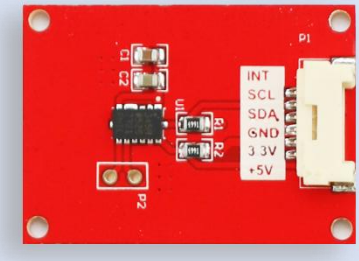
等待下一步的解决方案，目前暂时跳过。

## 5.2.4. 三轴加速度传感器（ADXL345）

三轴加速度使用 I2C 通信，模块如下：

表 5-6 三轴加速度模块

| 三轴加速度模块 | 概况 |
| --- | --- |




1 模块使用 ADXL345

2 使用 I2C 通信接口

3 可进行单振/双振检测，活动/非活动检测，自由落体检测

### 5.2.4.1. 实现原理及原理图简介

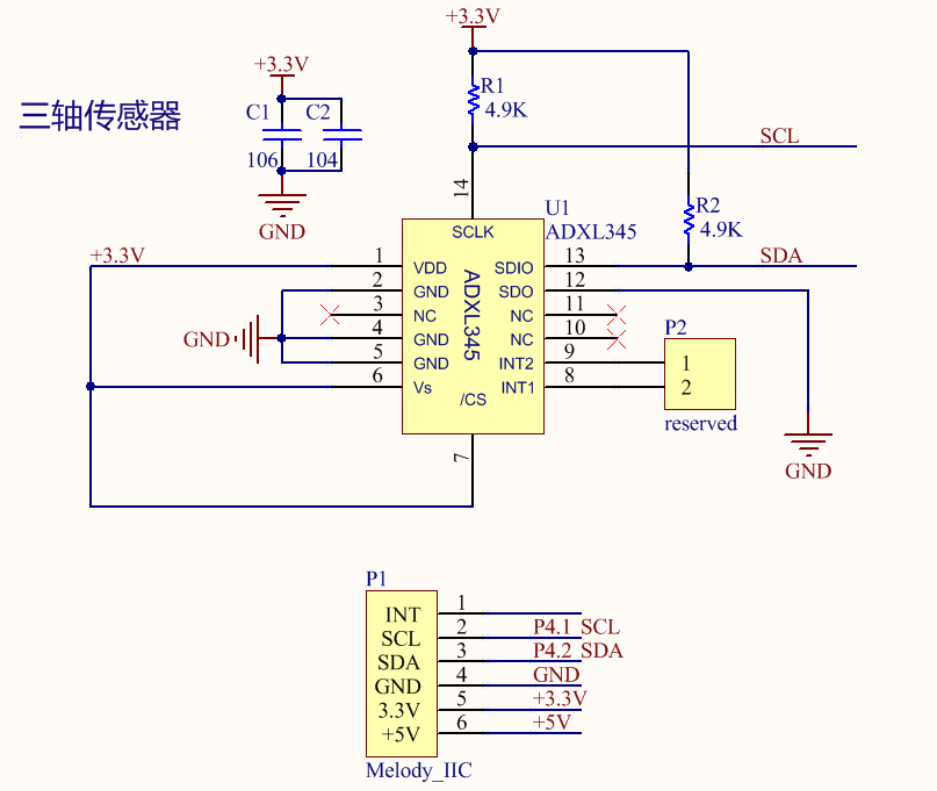



图 5-67 三轴加速度原理图

功能框图

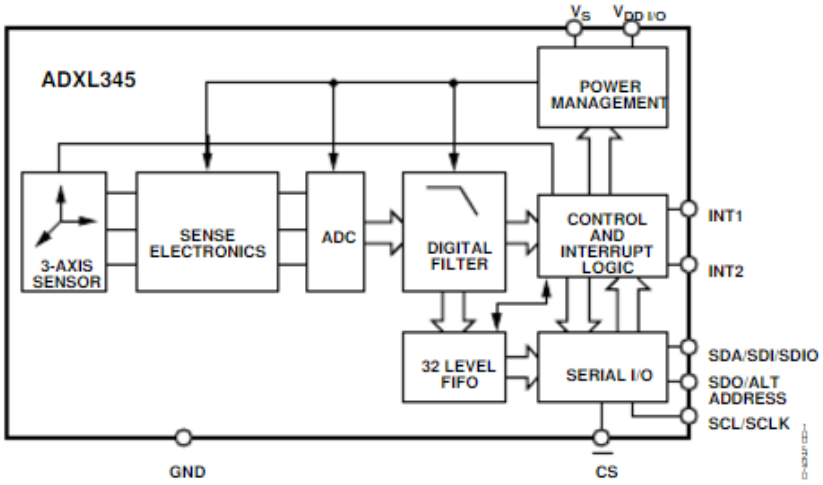



图 5-78　ADXL345 内部结构

ADXL345 是一款小而薄的超低功耗 3 轴加速度传感器，分辨率高（13 位），测量范围±16g，数字输出数据为 16 位二进制补码格式，可通过 I2C 接口进行访问。

### 5.2.4.2. 实现方法

➢ 编写驱动脚本：

三轴加速传感器的驱动需要熟悉 ADXL345 芯片基本原理和一系列寄存器的初始化。

```
classADXL345(I2C, Acceleration):
    def__init__(self, slave=0x53):
        I2C.__init__(self, toint(slave))
        self.writeRegister(0x31,0x0B)
        self.writeRegister(0x1E,0x00)#X 轴误差补偿; (15.6mg/LSB)
        self.writeRegister(0x1F,0x00)#Y 轴误差补偿; (15.6mg/LSB)
        self.writeRegister(0x20,0x00)#Z 轴误差补偿; (15.6mg/LSB)
        self.writeRegister(0x21,0x00)#敲击延时 0:禁用;(1.25ms/LSB)
        self.writeRegister(0x22,0x00)#检测第一次敲击后的延时 0:禁用;(1.25ms/LSB)
        self.writeRegister(0x23,0x00)#敲击窗口 0:禁用;(1.25ms/LSB)
        self.writeRegister(0x24,0x01)#保存检测活动阀值;(62.5mg/LSB)
        self.writeRegister(0x25,0x01)#保存检测静止阀值;(62.5mg/LSB)
        self.writeRegister(0x26,0x2B)#检测活动时间阀值;(1s/LSB)
        self.writeRegister(0x27,0x00)#
        self.writeRegister(0x28,0x09)#自由落体检测推荐阀值;(62.5mg/LSB)
        self.writeRegister(0x29,0xFF)#自由落体检测时间阀值,设置为最大时
间;(5ms/LSB)
        self.writeRegister(0x2A,0x80)#禁止敲击检测
        self.writeRegister(0x2C,0x08)#200hz 输出 0A 是 100hz 输出
        self.writeRegister(0x2D,0x08)#开启 Link,测量功能;关闭自动休眠,休眠,唤醒功能
```

```
        self.writeRegister(0x2E,0x80)#所有均关闭
        self.writeRegister(0x2F,0x00)#中断功能设定,不使用中断
        self.writeRegister(0x38,0x9F)#FIFO 模式设定,bypass 模式,31 级样本缓冲

    def __str__(self):
        return"ADXL345(slave=0x%02X)"% self.slave

    def __getAcceX__(self):#获取 X 轴加速度原始数据
        d0 = self.readRegisters(0x32,2)
        count =((d0[0]<<4)|(d0[1]>>4))&0xFFFF
        return signInteger(count,13)*0.000244140625
    def __getAcceY__(self):#获取 Y 轴加速度原始数据
        d1 = self.readRegisters(0x34,2)
        count =((d1[0]<<4)|(d1[1]>>4))&0xFFFF
        return signInteger(count,13)*0.000244140625
    def __getAcceZ__(self):#获取 Z 轴加速度原始数据
        d2 = self.readRegisters(0x36,2)
        count =((d2[0]<<4)|(d2[1]>>4))&0xFFFF
        return signInteger(count,13)*0.000244140625
```

- Webiopi 库添加驱动脚本

  在 WebIOPi-0.7.1\python\WebIOPi.egg-info\SOURCE.TXT 中添加：

  webiopi/devices/sensor/adxlXXX.py 一项

- 传感器管理中初始化驱动脚本

  在 WebIOPi-0.7.1\python\webiopi\devices\sensor__init__.py 中，设备管理项内添加 ADXL345；

  DRIVERS["adxlXXX"] = ["ADXL345"]

  同时加入新的 class 定义：

```
classAcceleration():
def __family__(self):
return"Acceleration"
def __getAcceX__(self):
raise NotImplementedError
def __getAcceY__(self):
raise NotImplementedError
def __getAcceZ__(self):
raise NotImplementedError
@request("GET","sensor/acceleration/gx")
@response("%.02f")
```

```
defgetAcceX(self):
return self.__getAcceX__()
@request("GET","sensor/acceleration/gy")
@response("%.02f")
defgetAcceY(self):
return self.__getAcceY__()
@request("GET","sensor/acceleration/gz")
@response("%.02f")
defgetAcceZ(self):
return self.__getAcceZ__()
```

- Webiopi.js 中添加新设备：

  在 addDevices 中也养添加新的加速度传感器类型选项：

```
    /*TODO ZK ADD Acceleration*/
    if(type =="Acceleration"){
        returnnew Acceleration(name);
    }
```

```
/**********************************************************************
*                    添加加速度传感器
*
**********************************************************************/
function Acceleration(name){
    this.name = name;
    this.url ="/devices/"+ name +"/sensor";
    this.refreshTime =5000;
}

Acceleration.prototype.toString=function(){
    returnthis.name +": Acceleration ";
}
Acceleration.prototype.getAcceX =function(callback){
    $.get(this.url +"/acceleration/gx",function(dataX){
        callback(this.name, dataX);
    });
}
Acceleration.prototype.getAcceY =function(callback){
    $.get(this.url +"/acceleration/gy",function(dataY){
        callback(this.name, dataY);
    });
}
Acceleration.prototype.getAcceZ =function(callback){
```

```
    $.get(this.url +"/acceleration/gz",function(dataZ){
        callback(this.name, dataZ);
    });
}
Acceleration.prototype.refreshUI =function(){
    var acce =this;
    varelement=this.element;
    if((element!=undefined)&&(element.header ==undefined)){
        element.header = $("<h3>"+this+"</h3>");
        element.append(element.header);
        acce.gx = $("<div id='gx'></div>");
        acce.gy = $("<div id='gy'></div>");
        acce.gz = $("<div id='gz'></div>");
        element.append(acce.gx);
        element.append(acce.gy);
        element.append(acce.gz);
    }
    this.getAcceX(function(name, dataX){
        if(element!=undefined){
            //element.header.text(acce + "X: " + dataX +  " G");
            acce.gx.text("Acceleration X: "+ dataX +" G");
        }
        setTimeout(function(){acce.refreshUI()}, acce.refreshTime);
    });
    this.getAcceY(function(name, dataY){
        if(element!=undefined){
            //element.header.text(acce + "Y: " + dataY + " G");
            acce.gy.text("Acceleration Y: "+ dataY +" G");
        }
        setTimeout(function(){acce.refreshUI()}, acce.refreshTime);
    });
    this.getAcceZ(function(name, dataZ){
        if(element!=undefined){
            //element.header.text(acce + "Z: " + dataZ + " G");
            acce.gz.text("Acceleration Z: "+ dataZ +" G");
        }
        setTimeout(function(){acce.refreshUI()}, acce.refreshTime);
    });
}
```

- Config 中创建设备

#加速度传感器

adxl = ADXL345

#### 5.2.4.3. 测试演示

将编辑的各种文档上传到 Raspberry 上对应的路径下重新编译 webiopi。完成后运行./runWebiopi.sh 脚本。在打印信息中我们能够看到如下 Debug 内容：

```
2017-03-23 07:47:43 - WebIOPi - INFO - Loading configuration from /etc/webiopi/config
2017-03-23 07:47:43 - WebIOPi - DEBUG - Mapping ADXL345(slave=0x53).getAcceX to REST GET /devices/adxl/sensor/acceleration/gx
2017-03-23 07:47:43 - WebIOPi - DEBUG - Mapping ADXL345(slave=0x53).getAcceY to REST GET /devices/adxl/sensor/acceleration/gy
2017-03-23 07:47:43 - WebIOPi - DEBUG - Mapping ADXL345(slave=0x53).getAcceZ to REST GET /devices/adxl/sensor/acceleration/gz
2017-03-23 07:47:43 - WebIOPi - INFO - Acceleration - ADXL345(slave=0x53) mapped to REST API /devices/adxl
2017-03-23 07:47:43 - WebIOPi - INFO - Access protected using /etc/webiopi/passwd
```

图 5-89 运行 webiopi

意义就在于创建了 Restful，可以供 web 端不断的 get 到三轴加速度传感器的数据更 gx,gy,gz 三轴加速度数据值。打开网页点击 Devices Monitor：

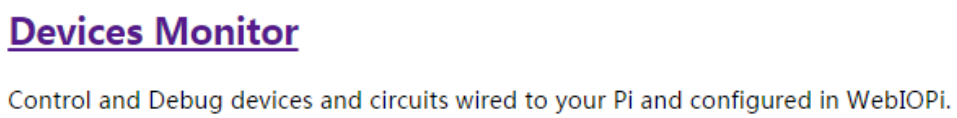



图 5-40 进入设备管理链接

可以看到如下的显示内容：

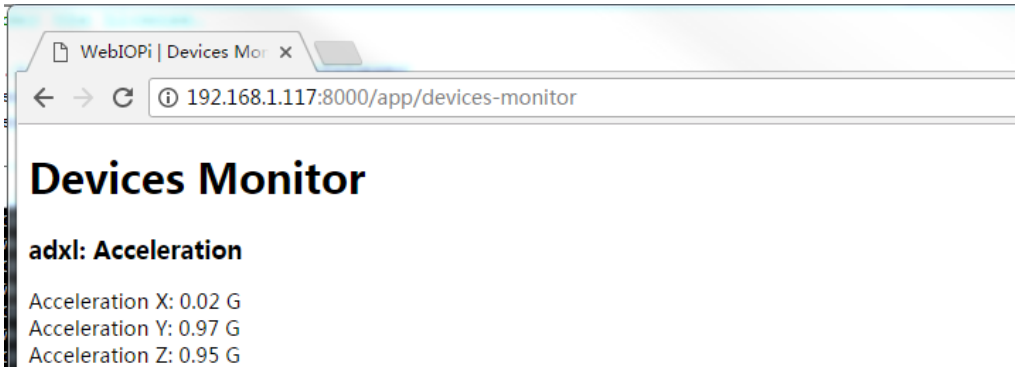



图 5-41 显示三轴加速度数据

这样我们就能够通过移动端获取到挂载在树莓派上的任何传感器的数据了。

2017-03-23 07:49:30 - HTTP - DEBUG - "GET /devices/adxl/sensor/acceleration/gx HTTP/1.1" - 200 OK (Client: ::ffff:192.168.1.18 <Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.14 Safari/537.36>)

2017-03-23 07:49:30 - HTTP - DEBUG - "GET /devices/adxl/sensor/acceleration/gy HTTP/1.1" - 200 OK (Client: ::ffff:192.168.1.18 <Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.14 Safari/537.36>)

2017-03-23 07:49:30 - HTTP - DEBUG - "GET /devices/adxl/sensor/acceleration/gz HTTP/1.1" - 200 OK (Client: ::ffff:192.168.1.18 <Mozilla/5.0 (Windows NT 6.1; WOW64)

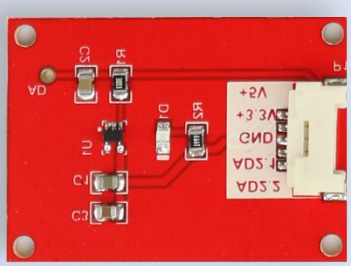
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.14 Safari/537.36>)

查看后台打印信息我们能够看到，weiiopi 服务会间隔的想 http 提交 /devices/adxl/sensor/acceleration/gx 等加速度数据。假如我们停留在 Devices Monitor 页面，后台会间隔 5 秒不断地更新加速度传感器的数据。假如退出来网页，则更新停止。

## 5.2.5. AD 采样驱动（LMT84）

模拟温度模块使用 AD 接口进行温度采集。模块简介如下：

表 5-7 模拟温度采集模块

| 模拟温度采集模块 | 概况 |
| --- | --- |
|  | 模块使用芯片 LMT84<br>使用 ADC 接口采集温度数据<br>测量精度 ±0.4℃<br>输出受到短路保护 |

### 5.2.5.1. 实现原理及原理图简介

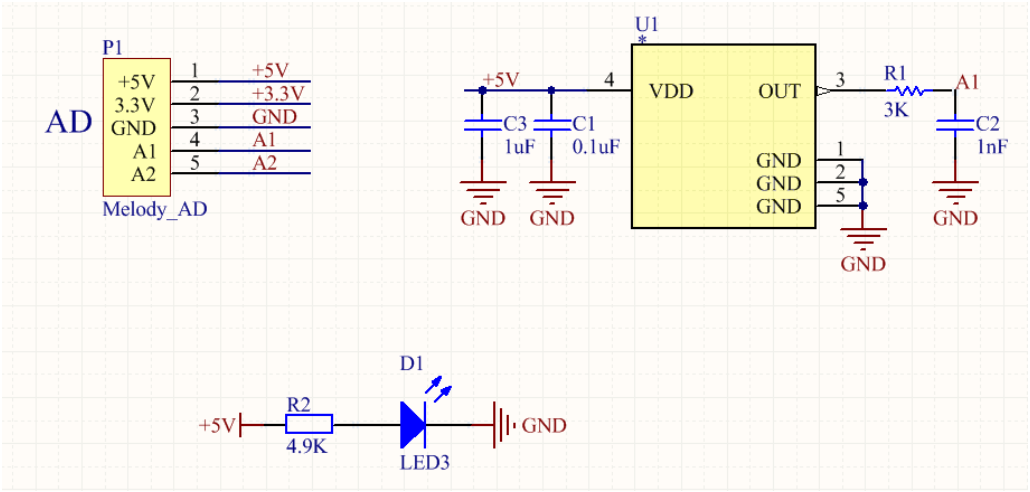



图 5-42 模拟温度模块原理图

LMT84 是高精度 CMOS 集成电路温度传感器，此传感器具有与温度成线性反比例关系的模拟电压输出，即可获得 LMT84 的模拟电压输出，进而转换成温度。测量范围为-50℃~150℃。

测量温度与模拟电压关系为：T=（Vout-1035mV）/（-5.5mV/℃）

详细的对应关系，可参考 LMT84 的手册

http://www.ti.com.cn/cn/lit/ds/symlink/lmt84.pdf

需要关注的是，树莓派是不带模拟量输入的，所以我们这里实际使用的是一块 ads1100 中间转换电路将模拟量转换为数字量并通过 I2C 总线传输给 Raspberry。

#### 5.2.5.2. 实现方法

- 编写驱动脚本：

由于使用的实际是转接板上的 ADS1100 的 I2C 通信功能，实际上它的脚本编写的就是 I2C 驱动。所以和 OPT3001 以及温湿度的类似：

```
from webiopi.devices.i2c import I2C
from webiopi.devices.sensor import Temperature
classLMT84(I2C, Temperature):
    def__init__(self, slave=0x48):
        I2C.__init__(self, toint(slave))
    def__str__(self):
        return"LMT84(slave=0x%02X)"% self.slave
    def__getCelsius__(self):
        d0 = self.readRegisters(0x0,2)
        count =((d0[0]<<4)|(d0[1]>>4))&0xFFFF
        return signInteger(count,13)
```

编写完成后，保存为 lmtXX.py 保存在

WebIOPi-0.7.1\python\webiopi\devices\sensor 文件夹下。

- Webiopi 库添加驱动脚本

在 WebIOPi-0.7.1\python\WebIOPi.egg-info\SOURCE.TXT 中添加：

webiopi/devices/sensor/lmtXX.py 一项源码关联。

- 传感器管理中初始化驱动脚本

在 WebIOPi-0.7.1\python\webiopi\devices\sensor__init__.py 中，设备管理项内添加 LMT84；

DRIVERS["lmtXX"] = ["LMT84"]

同时加入 class 定义：由于使用的还是 Temperature 的定义，所以在这里我们无需增加。

- Webiopi.js 中添加新设备

  同上，采用和 Temperature 同样的配置，这里无需增加内容。

- Config 中创建设备

  在[DEVICES]字段下新增加一个设备名：

  #AD ADS1100 转换

  ad = LMT84

#### 5.2.5.3. 测试演示

将编辑完成的各种文档上传到 Raspberry 上对应的路径下重新编译 webiopi。完成后运行./runWebiopi.sh 脚本。在打印信息中我们能够看到如下 Debug 内容：

```
2017-03-24 01:47:25 - WebIOPi - INFO - Loading configuration from /etc/webiopi/config
2017-03-24 01:47:25 - WebIOPi - DEBUG - Mapping LMT84(slave=0x48).getCelsius to REST GET /devices/ad/sensor/temperature/c
2017-03-24 01:47:25 - WebIOPi - DEBUG - Mapping LMT84(slave=0x48).getFahrenheit to REST GET /devices/ad/sensor/temperature/f
2017-03-24 01:47:25 - WebIOPi - DEBUG - Mapping LMT84(slave=0x48).getKelvin to REST GET /devices/ad/sensor/temperature/k
2017-03-24 01:47:25 - WebIOPi - INFO - Temperature - LMT84(slave=0x48) mapped to REST API /devices/ad
```

图 5-43 运行 webiopi

意义就在于创建了 Restful，可以供 web 端不断的 get 到 LMT84 的温度数据。打开网页点击 Devices Monitor：

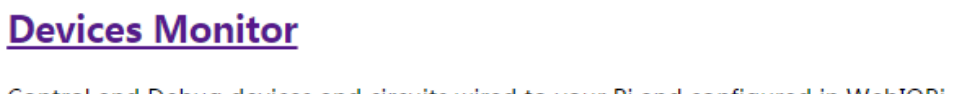



图 5-44 进入设备管理链接

可以看到如下的显示内容：


图 5-45 查看到 AD 的数据

这样我们就能够通过移动端获取到挂载在树莓派上的任何传感器的数据了。

2017-03-24 01:49:58 - HTTP - DEBUG - "GET /devices/ad/sensor/temperature/c HTTP/1.1" - 200 OK (Client: ::ffff:192.168.1.18 <Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.14 Safari/537.36>)

查看后台打印信息我们能够看到，weiiopi 服务会间隔的想 http 提交

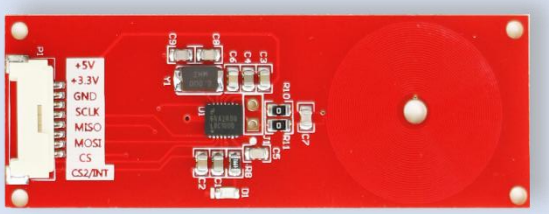
/devices/ad/sensor/temperature/c 温度数据。假如我们停留在 Devices Monitor 页面，后台会间隔 5 秒不断地更新传感器的数据。假如退出来网页，则更新停止。

## 5.3. SPI 通信

### 5.3.1. 接近传感器

LDC1000 使用 SPI 通信，模块如下

表 5-8 LDC1000 模块

| LDC1000 模块 | 概况 |
| --- | --- |
|  | 1、SPI 接口通信<br>2、可对线性位置或者角位置、位移、运动、压缩、振动以及金属成分进行测量 |

### 5.3.1.1. 实现原理及原理图简介

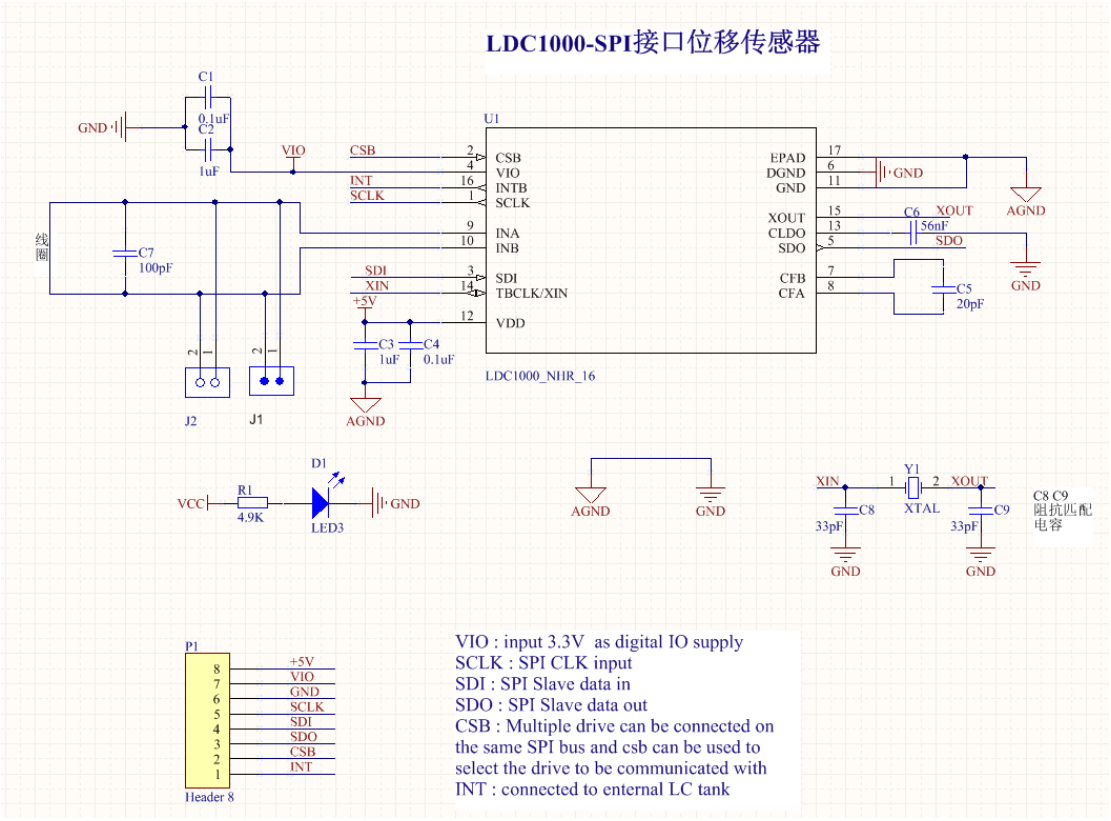



图 5-46 LDC1000 模块原理图

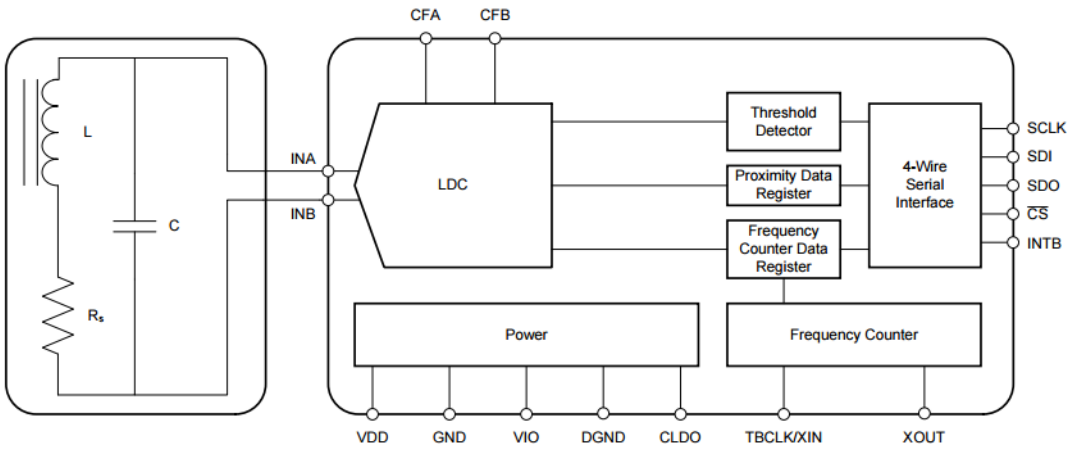



图 5-47 LDC1000 功能结构图

LDC1000 是世界首款电感到数字转换器。提供低功耗，小封装，低成本解决方案。SPI 接口很方便连接 Raspberry。LDC1000 只需要外接一个 PCB 线圈或者自制线圈就可以实现非接触式电感检测。LDC1000 的电感检测并不是指像 Q 表那样测试线圈的电感量，而是可以测试外部金属物体和 LDC1000 的测试线圈的空间位置关系。

利用 LDC1000 这个特性配以外部设计的金属物体即可很方便实现：水平或垂

直距离检测；角度检测；位移监测；运动检测；振动检测；金属成分检测（合金检测）。可以广泛的应用在汽车、消费电子、计算机、工业、通信和医疗领域。LDC1000 测量电感频率是用测试 LC 谐振频率的方法。LDC1000 有外部的基准时钟，也是使用计数方法来做频率计。看一下手册中给出的公式就能清晰的看出计数原理。

$$f_{sensor} = (1/3) \times (F_{ext} / F_{count}) \times (responseTime)$$

fsenser 是 LC 谐振频率，Fext 是外部基准时钟频率，Fcount 是 LDC1000 内部计数器值（0x23,0x24,0x25），Response time 是寄存器设定的一个值（0x04）。将上面公式左右分别求倒数

$$1/f_{sensor} = 3 \times (F_{ext} / F_{count}) \times (responseTime)$$

将 response time 移动到等式左面

$$responseTime \times (1/f_{sensor}) = 3 \times F_{count} \times (1/F_{ext})$$

这样看就很清晰了，1/fsensor 是 LC 谐振周期，1/Fext 是基准时钟周期，也就是说在 response time 个 LC 谐振周期内，使用 LDC1000 的 Fcount 计数器记录基准时钟的个数用来推算 LC 的谐振频率。根据 fsensor= 1/ 计算出 L。（C 是电路设计中的已知量）

$$L = 1/[C \times (2 \times \pi \times f_{sensor})^2]$$

#### 5.3.1.2. 实现方法

未完待续！

#### 5.3.1.3. 测试演示

未完待续！

## 5.4. 总结

艾研信息的物联网套件总体而言是一款主要面向高校物联网课程教学开发的实验套件。能够让学生在实验和实践的过程中理解和体会物联网开发的整个框架和流程。