
# 物联网服务器的开发与集成

**本章导读**

第4章实现了物联网的“前端”开发与树莓派的集成，在树莓派的显示屏上，不但可以看见比数码管更美观的温度显示，而且用户还可以通过可视化的界面（控制面板），设置报警、反向控制参数的阈值，以及自动和手动进行反向控制。这是一个传统自动控制系统的“基本雏形”，一个真实的自动控制系统可以在这个基础上进行扩展，达到真正实用性要求，这也是本章第3节的任务。

本章的前两小节课程介绍如何将第4章完成的前端系统集成到物联网服务器上，实现用手机控制“前端”设备。5.1节介绍利用别人（Yeelink）的服务器，实现“既定”的物联网功能，5.2节介绍如何在“苏宁云”上，完全从“零”（操作系统环境）开始，搭建自己的物联网应用服务器，体验物联网应用层的开发与集成。5.3节以电梯群控为目标，将之前已经实现的小型控制系统，放到电梯群控的实际背景下，探讨担负不同目标任务的三个子系统之间，任务责任如何划分、信息如何交互、运行如何协调，并最终满足电梯群控“分布、并发、实时”的需求。5.4节讨论如何以物联网大赛为目标，设计自己的参赛项目。

现在的自动控制，早已不是各级监控人员坐在各自监控室的“控制面板”前，眼睛盯着监控屏幕，发出控制指令，进行监控的情形。监控系统大多采取“大集中”、“远程”、“自动识别、记录、报警”、智能动作方式进行监控，人工只是起辅助和关键决策作用。例如，国内银行界早在十多年前（国税、电信，甚至公安系统也已开始分步实施）已逐渐开始实施并完成了全行业务系统的“大集中”，全行所有业务软件系统，以及机房设备、关键工作场所、所有交易的柜员和客户操作、自动无人环境操作（如ATM机）等，都处于远程监控之下。而从2014年开始，从ATM机里吐出的每张人民币的序列号，都是处于监控状态（在ATM机的出钞口进行自动识别、记录在案）的。自从2005年伦敦地铁爆炸案开始，大街上装满了各个安保单位、各种用途的摄像头，物联网在智慧城市的建设中，首先在确保社会治安方面发挥了重要的作用。

“大集中”模式对数据的采集、处理、应用，不受对象、时间、地点、位置、状态的限制，更克服了“分级式”系统因人为因素所造成的应用系统的差异性，在信息收集上，因层层上报而导致的信息丢失、人为过滤、隐瞒、延误等问题。同时，大大降低了整个系统在系统开发、设备维护、人员使用方面的直接成本和管理成本。

从技术实现角度看，“大集中”模式，在应用层，无一不是借助“互联网”，并在互联网的

基础上，实现“分布数据采集”与集中数据处理与应用相结合的系统模式。目前，实现这种模式的应用系统，从主机和存储、数据传输、应用的响应速度等方面，已经完全不是问题。某些领域应该而暂时还没有完全实现“大集中”的原因，基本都不是技术性的原因。可以预计：凡是“国”字号的行业系统，早晚是全国规模的“大集中”，全国，甚至跨国性的企业也是一样。这是本章物联网应用的大背景。

从下节开始，尝试先把树莓派上获得的温度数据送到网上，让用户可以看到这些数据。

## 5.1 与 Yeelink 服务器集成

Web 应用开发是一门专业课程，包括 Web 前端开发，涉及 HTML5、CSS3、JavaScript 以及 iOS、Android 两大平台开发技术等。Web 服务器端开发，涉及 XML、Servlet、SSH 架构、数据库等技术。这些技术，是当前物联网应用层的开发中比较流行的技术方法。由于本课程不是专门介绍这些技术的课程，只是运用这些技术，因此，在涉及相关技术时，不太熟悉的同学，应尽量去查找相关技术资料，补习相关技术知识。

### 5.1.1 物联网服务器 Yeelink 的功能与局限

根据从简单到复杂、先易后难的原则，本小节先借用一个现成的物联网服务器，通过与服务器简单的数据传输和下载，实现一个最基本的传感器数据采集和监控功能，然后再探讨如何开发和集成更复杂的物联网应用。

为了达到上述目的，选择了 Yeelink 服务器作为物联网应用服务器。Yeelink 具有以下功能特点。

(1) 接入与存储：Yeelink 是一个具有支持高并发接入和云存储的物联网应用服务器。所谓高并发接入是指它能够同时(并发)完成海量的传感器数据接入并存储在 Yeelink 的云服务器上，并确保数据存储是安全的。

(2) 事件触发机制：当数据达到某个设定阈值的时候，Yeelink 平台会自动调用预先设定的规则，发送短信、微博，或者是邮件。作为对传感器数据的“应用”，Yeelink 能做到的“数据处理和应用”，就是当某个数值达到设定的阈值的时候，发送短信或邮件而已。如果要想在数据“处理和应用”方面再进一步，例如当发生某些“事件”时，运行自己定义的某些“动作”，如运行某个自己编写的进程程序，Yeelink 是不允许的，如果可以，Yeelink 就是一个将在下一节讨论的“私有云”服务器了。

(3) 数据展现：Yeelink 上所有的数据，均可以通过在 IE 上，按地图和时间轴方式进行展现，也可以使用 iphone 或 Android 手机上的 APP，很容易地下载 Yeelink 上被公开的传感器，获取诸如空气质量、PM2.5 指数等数据，停车场的剩余车位数量，或是获取其他类似公交状况等城市公共数据。当然，Yeelink 的数据，如 PM2.5、停车位置等，全部是自己采集、上传、下载的，任何数据的含义、使用，也由自己定义。Yeelink 平台只把它们看成是单纯的“数据”，所以，它只是一个数据承载平台。

(4) 双向传输和控制功能：Yeelink 平台的最大特点，在于不仅仅能够提供数据的上

行功能，还可以通过 APP 等接口，实现数据的下行功能。通过下行数据，能够实现诸如对家庭电器的控制等。例如，快要到家前想洗个热水澡，或是要提前把空调打开。反向控制可以实现这些很简单的要求。但是，仅用手机上的 APP，还是不能真正打开热水器、启动空调，这些“举手之劳”的简单操作，即使控制命令到达了放在家里的手机上，离热水器、空调还有“最后一米”的距离，这是需要借助家用电器“智能化”来做的事情，Yeelink 也是实现不了的。

（5）社交网络融合：在 Yeelink 上存储的数据，可以简单地被手机 API 取回。在手机上编写一个应用读取 Yeelink 上的数据，并嵌入到个人博客上，或者根据规则自动转发到指定的微博上，实现传感器数据与应用、与人之间的全面融合。注意：Yeelink 只提供 APP 接口，应用程序还是需要你自己写的。

至此，大致了解了 Yeelink 可以做到什么、做不到什么。相信 Yeelink 也会不断地发展，提供越来越强大的物联网应用服务功能。但 Yeelink 不论怎么发展，也一定会有一个“度”，就是什么是数据承载平台、什么是数据“应用”服务器、边界在哪里。

Yeelink 是国内比较著名的传感器数据开放平台，这个领域的带头者，可能是 2009 年的 Pachube（2011 年 7 月被云端计算机服务供应商 LogMeIn 收购）。类似、免费开放的网站还有不少，并不是只有 Yeelink 一家，所实现的功能也基本相似。

本节先体验 Yeelink 可以做到的部分，下一节继续开发 Yeelink 也做不到的功能。

### 5.1.2 Yeelink 的接入设备与存储数据类型

作为物联网的应用服务器，Yeelink 宣称可以接入任何类型的传感器设备。在 Yeelink 上创建自己的设备的时候，Yeelink 允许选择的设备类型如图 5-1 所示。


图 5-1　Yeelink 允许的接入设备类型

前几个设备类型比较容易理解，包括数值型传感器（数据值）、开关（状态）、GPS（位置值）、图像（静态图片或作为图片看待的动态图像）。

泛传感器是 Yeelink 一个新扩展的功能，也是其创造的一个新概念术语，也就是所谓的通用传感器（Generic Sensor）。以前 Yeelink 甚至 Pachube 支持的传感器都是数值型的，没法表达更复杂的结构化数据。所谓复杂数据，例如 GPS 数据，其一条数据就包含经纬度、海拔、速度、航向等很多信息。对于更复杂的设备，或者对于控制来说，简单的数值型数据无法满足应用的需求。现在 Yeelink 的泛传感器就是为了解决复杂的结构化数据交互的需求，而新增的数据类型。

当在用户中心增加一个泛传感器时，页面上看到的是如下的传感器介绍：“这是一个泛传感器，它支持上传任意形式的数据，用户可自定义具体内容”。该功能支持的数据采用 JSON 格式，使用 key-value 方式进行存取操作。

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式。它基于JavaScript的对象表示方法的一个子集。其本质上,就是一种单个数据或数组的组织形式,以及与此相匹配的数据操作方式。JSON采用完全独立于语言的文本格式,但是也使用了类似于C语言家族的习惯(包括C、C++、C#、Java、JavaScript、Perl、Python等),这些特性使JSON成为理想的数据交换语言。易于人阅读和编写,同时也易于机器解析和生成(网络传输速率)。有关JSON的数据格式,将在"树莓派与物联网服务器通信"一节中介绍。

泛型传感器的一个具体的例子就是GPS。由于GPS有着广泛而特殊的应用领域,在Yeelink系统中,又把GPS单独作为一种数据类型和泛传感器并列,但在数据存储和操作上,GPS的本质就是上述的JSON类型。

泛传感器类型的应用举例:RGB LED控制。

如果希望做一个更炫的RGB LED远程控制,与过去相比,现在需要用到泛传感器去传递结构化的颜色分量信息,例如一组数据形式为:

```
{
red: 255,
blue: 0,
green: 0
}
```

这个例子说明,泛传感器的数据是一个由多个"数值对(red、blue、green)"组成的数组。

但是,从Yeelink可以提供的应用功能(接口和手机APP)"反推",Yeelink所能接入、存储的传感器数据,必定只能是一些"数值",而Yeelink就是在"云"上的数值寄存器,最多是一个"数组"而已。

图5-1中,Yeelink的最后一个数据类型是"微博抓取器"。这是什么东西呢?同学们自己尝试一下。

### 5.1.3 在Yeelink上创建自己的设备和传感器

要想使用Yeelink作为传感器数据服务器,需要先在Yeelink上注册并添加自己的传感器设备。目前Yeelink还是免费使用的。

#### 1. 在Yeelink上注册

访问Yeelink的官方网站(http://www.yeelink.net/),单击"快速开始",按照步骤注册账号。

#### 2. 添加一个设备和传感器

Yeelink把上传数据划分为"设备"和"传感器"两大类。图5-1显示了允许添加的设备类型,在一个类型的"设备"之下,可以有多个该类型的传感器。例如,"温度"是一个数值型的设备,温度1、温度2、……是$N$个温度传感器,当然也是数值型。

按照 Yeelink 的官方向导，添加设备和传感器的过程是：(1)进入用户中心；(2)在“我的设备”中，选择增加新设备。在添加设备过程中，根据设备类型，选择合适的数据类型。例如：是开关设备(传感器本身具有 0 或者 1 的物理属性)，还是数值型设备(温度、气压、亮度、湿度等)，或者是图像数据(摄像机)等，也可以是上节介绍的泛传感器(一组成对的数据)或微博 URL。添加成功后，再在设备之下添加传感器。不要添加了设备就以为添加结束了。这里，“设备”类似 C++ 中的类，而“传感器”才是真正的对象。但类定义了对象的性质——数据类型。添加成功后，可以单击“设备管理”，看见自己新添加的设备和设备下的传感器，如图 5-2 所示。


图 5-2　添加设备后获得设备/传感器编号

要记住 Yeelink 分配给这个设备的设备编号和传感器编号，这是用户上传数据的目标地址，也是以后在 Yeelink 上搜索设备的依据(而不是用户名)。在图 5-2 的示例中，设备号是 URL：http//api. yeelink. net/v1. 0/device/340581/sensor/377263/datapoints 中的 340581(设备号)和 377263(传感器号)两个数字，其他字符都是“标准”的，照抄就行。

### 5.1.4　HTTP 协议简介

HTTP 是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。它于 1990 年提出，经过几十年的使用与发展，得到不断的完善和扩展。目前在 WWW 中使用的是 HTTP/1.0 的第 6 版。

#### 1. HTTP 协议的主要特点

(1) 支持客户/服务器模式。

(2) 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。

(3) 灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。

(4) 无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

(5) 无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

在上述介绍中，有些名词可能暂时难于理解，例如连接、状态等，这是计算机通信的基本概念，请参考有关网络通信的相关内容。

### 2. HTTP 的 URL

HTTP URL 是一种特殊类型的 URI，包含了用于查找某个资源的足够的信息，其格式如下：http://host[":"port][abs_path]。http 表示要通过 HTTP 协议来定位网络资源；host 表示合法的 Internet 主机域名或者 IP 地址；port 指定一个端口号，为空则使用缺省端口 80；abs_path 指定请求资源的 URI；如果 URL 中没有给出 abs_path，那么当它作为请求 URI 时，必须以“/”的形式给出，通常这个工作浏览器自动完成。例如输入“www.guet.edu.cn”，浏览器自动转换成“http://www.guet.edu.cn/”。

### 3. HTTP 的请求

HTTP 的请求由三部分组成，分别是请求行、消息报头、请求正文。

(1) 请求行以一个方法符号开头，以空格分开，后面跟着请求的 URI 和协议的版本，格式如下：Method Request-URI HTTP-Version CRLF。其中 Method 表示请求方法；Request-URI 是一个统一资源标识符；HTTP-Version 表示请求的 HTTP 协议版本；CRLF 表示回车和换行(除了作为结尾的 CRLF 外，不允许出现单独的 CR 或 LF 字符)。

请求方法(所有方法全为大写)有多种，各个方法的解释如下。

① GET：请求获取 Request-URI 所标识的资源；

② POST：在 Request-URI 所标识的资源后附加新的数据；

③ HEAD：请求获取由 Request-URI 所标识的资源的响应消息报头；

④ PUT：请求服务器存储一个资源，并用 Request-URI 作为其标识；

⑤ DELETE：请求服务器删除 Request-URI 所标识的资源；

⑥ TRACE：请求服务器回送收到的请求信息，主要用于测试或诊断；

⑦ CONNECT：保留将来使用；

⑧ OPTIONS：请求查询服务器的性能，或者查询与资源相关的选项和需求。

应用举例：

GET 方法：以在浏览器的地址栏中输入网址的方式访问网页时，浏览器采用 GET 方法向服务器获取资源，例如 GET /form.html HTTP/1.1(CRLF)。

POST 方法要求被请求服务器接收附在请求后面的数据，常用于提交表单。

例如：

```
POST /reg.jsp HTTP/ (CRLF):
Accept:image/gif,image/x-xbit,...(CRLF)
...
HOST:www.guet.edu.cn (CRLF)
Content-Length:22 (CRLF)
Connection:Keep-Alive (CRLF)
Cache-Control:no-cache (CRLF)
```

```
(CRLF)                          //该 CRLF 表示消息报头已经结束,在此之前为消息报头
user=jeffrey&pwd=1234           //此行以下为提交的数据
```

HEAD 方法与 GET 方法几乎是一样的,对于 HEAD 请求的回应部分来说,它的 HTTP 头部中包含的信息与通过 GET 请求所得到的信息是相同的。利用这个方法,不必传输整个资源内容,就可以得到 Request-URI 所标识的资源的信息。该方法常用于测试超链接的有效性、是否可以访问,以及最近是否更新。

(2) 请求报头后述。

(3) 请求正文略。

#### 4. HTTP 的响应

在接收和解释请求消息后,服务器返回一个 HTTP 响应消息。HTTP 响应也是由三个部分组成的,分别是状态行、消息报头、响应正文。

#### 5. HTTP 的报头

HTTP 消息由客户端到服务器的请求和服务器到客户端的响应组成。请求消息和响应消息都是由开始行(对于请求消息,开始行就是请求行;对于响应消息,开始行就是状态行),消息报头(可选),空行(只有 CRLF 的行),消息正文(可选)组成的。

HTTP 消息报头包括普通报头、请求报头、响应报头、实体报头。每一个报头域都是由名字+“:”+空格+值组成的,消息报头域的名字是大小写无关的。

在普通报头中,有少数报头域用于所有的请求和响应消息,但并不用于被传输的实体,只用于传输的消息。请求报头允许客户端向服务器端传递请求的附加信息以及客户端自身的信息。响应报头允许服务器传递不能放在状态行中的附加响应信息,以及关于服务器的信息和对 Request-URI 所标识的资源进行下一步访问的信息。请求和响应消息都可以传送一个实体。一个实体由实体报头域和实体正文组成,但并不是说实体报头域和实体正文要在一起发送,可以只发送实体报头域。实体报头定义了关于实体正文(例如有无实体正文)和请求所标识的资源的元信息。

### 5.1.5 有关 Requests

为了实现用程序向 Yeelink 上传数据,首先介绍需要在树莓派上导入的两个库:Requests 和 JSON。本节介绍 Requests 库及其使用方法。

Requests 库是用 Python 语言编写,基于 urllib 库,采用 Apache2 Licensed 开源协议的 HTTP 库。它比 urllib 库更加方便,可以节约大量的网络编程工作,完全满足上述的 HTTP 协议规定。

1) 安装 Requests

可以通过 pip 工具进行安装,如果树莓派上没有安装过 pip,可以用 apt-get 先安装 pip。然后执行:

```
$sudo pip install requests
```

也可以直接下载代码后安装，执行：

```
$git clone git://github.com/kennethreitz/requests.git
$cd requests
$python setup.py install
```

2）发送请求与传递参数的实现

```
import requests
r=requests.get(url='http://www.itwhy.org')          #最基本的 GET 请求
print(r.status_code)                                #获取返回状态
r=requests.get(url='http://dict.baidu.com/s', params={'wd':'python'})
                                                    #带参数的 GET 请求
print(r.url)
print(r.text)                                       #打印解码后的返回数据
```

从上例中可以看到，requests 使用的是 HTTP 协议中的 GET 方式，进行数据请求，并获得响应。当然也可以用其他请求方式，如：

```
requests.get('https://github.com/timeline.json')    #GET 请求
requests.post("http://httpbin.org/post")            #POST 请求
requests.put("http://httpbin.org/put")              #PUT 请求
requests.delete("http://httpbin.org/delete")        #DELETE 请求
requests.head("http://httpbin.org/get")             #HEAD 请求
requests.options("http://httpbin.org/get")          #OPTIONS 请求
```

另外，请求也是可以带参数的，实例代码如下：

```
import requests
requests.get('http://www.dict.baidu.com/s', params={'wd': 'python'})
                                                    #GET 参数实例
requests.post('http://www.itwhy.org/wp-comments-post.php', data={'comment':
'测试 POST'})                                       #POST 参数实例
```

发送数据的代码如下：

```
import requests
import json
r=requests.post('https://api.github.com/some/endpoint', data=json.dumps
({'some': 'data'}))
print(r.json())
```

在这段代码中，发送数据的格式采用的是 JSON 格式，下面会介绍。

更进一步，“报头”也可以定制，实例代码如下：

```
import requests
import json
```

```
data={'some': 'data'}
headers={'content-type': 'application/json',
                                        //指明实体正文数据类型为 application/json
'User-Agent': 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:22.0) Gecko/20100101
Firefox/22.0'}                          //OS 系统信息等
r=requests.post('https://api.github.com/some/endpoint', data=data, headers=
headers)
print(r.text)
```

3）接收响应对象

服务器响应 requests 请求后，会返回一个 response 对象，其存储了服务器响应的内容，如上实例中已经提到的 r.text、r.status_code 等。

获取文本方式的响应体实例：访问 r.text 之时，会使用其响应的文本编码进行解码，并且可以修改其编码让 r.text 使用自定义的编码进行解码。

```
r=requests.get('http://www.itwhy.org')
print(r.text, '\n{}\n'.format('*'*79), r.encoding)
r.encoding='GBK'
print(r.text, '\n{}\n'.format('*'*79), r.encoding)
```

其他的响应形式还包括：

```
r.status_code #响应状态码;
r.raw #返回原始响应体,也就是 urllib 的 response 对象,使用 r.raw.read()读取;
r.content #字节方式的响应体,会自动解码 gzip 和 deflate 压缩;
r.text #字符串方式的响应体,会自动根据响应头部的字符编码进行解码;
r.headers #以字典对象存储服务器响应头,但是这个字典比较特殊,字典键不区分大小写,若键
不存在则返回 None。
#*特殊方法*#
r.json() #Requests 中内置的 JSON 解码器;
r.raise_for_status() #失败请求(非 200 响应)抛出异常。
```

4）身份验证

身份认证方法是基于 Auth 的（HTTP Basic Auth），验证代码如下：

```
import requests
from requests.auth import HTTPBasicAuth
r=requests.get('https://httpbin.org/hidden-basic-auth/user/passwd', auth=
HTTPBasicAuth('user', 'passwd'))
#r=requests.get('https://httpbin.org/hidden-basic-auth/user/passwd', auth=
('user', 'passwd'))    #简写
print(r.json())
```

另一种非常流行的 HTTP 身份认证形式是摘要式身份认证，Requests 对它的支持也是开箱即可用的：

```
requests.get(URL, auth=HTTPDigestAuth('user', 'pass'))
```

### 5.1.6 有关 JSON 库

1) JSON 数据结构

JSON 建构于以下两种结构。

1)“名称/值”对

“名称/值”对组合中的名称写在前面(在双引号中),值对写在后面(同样在双引号中),中间用冒号隔开。一个“名称/值”对的例子是"firstName":"John"。这很容易理解,该“名称/值”对等价于这条 JavaScript 语句:firstName="John"JSON。

JSON 的值可以是数字(整数或浮点数)、字符串(在双引号中)、逻辑值(true 或 false)、数组(在方括号中)、对象(在花括号中)、null。

JSON 的“名称/值”对可以看成是 JavaScript 语句中的对象。对象在 js 中表示为用{}括起来的内容,数据结构为 {key: value,key: value,...}的键值对的结构,在面向对象的语言中,key 为对象的属性,value 为对应的属性值,所以很容易理解,取值方法为:对象.key 获取属性值,这个属性值的类型可以是数字、字符串、数组、对象等几种。

2) JSON 数组

JSON 数据可以是一个或多个上述对象构成的数组,数组在 js 中是中括号[]括起来的内容,数据结构为 ["java","javascript","vb",...],取值方式和所有语言中一样,使用索引获取,字段值的类型可以是数字、字符串、数组、对象等几种。

经过对象、数组两种结构,就可以组合成复杂的数据结构了。

在不同的语言中,JSON 的数组可能被理解为对象(object)、记录(record)、结构(struct)、字典(dictionary)、哈希表(hash table)、有键列表(keyed list),或者关联数组(associative array)、值的有序列表(an ordered list of values)等不同的概念,但本质是一样的。而最容易的理解还是数组(array)。

2. JSON 库与 JSON 数据处理

JSON 库封装了对 JSON 数据的 encoding 和 decoding 处理。使用简单的 json.dumps 方法,可以对 JSON 数据类型进行编码,例如:

```
import json
obj=[[1,2,3],123,123.123,'abc',{'key1':(1,2,3),'key2':(4,5,6)}]
encodedjson=json.dumps(obj)          //encoding 处理
print repr(obj)                      //显示原始数据
print encodedjson                    //encoding 显示处理后的数据
```

输出结果是:

```
print repr(obj) 输出: [[1, 2, 3], 123, 123.123, 'abc', {'key2': (4, 5, 6), 'key1':
(1, 2, 3)}]
print encodedjson 输出: [[1, 2, 3], 123, 123.123, "abc", {"key2": [4, 5, 6],
"key1": [1, 2, 3]}]
```