第五章 物联网服务器的开发与集成

目录

本章导读

上一章实现了物联网的“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、反向控制参数的阀值、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，介绍如何将上一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。前一节介绍利用别人（YeeLink）的服务器，实现“既定”的物联网功能，后一节介绍如何在“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。第三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。最后一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，早已不是各级的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 10 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，在涉及到相关技术时，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：

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，并确保你的数据存储是安全的。

事件触发机制：当你的数据达到某个你自己设定阈值的时候，Yeelink 平台会自动调用你预先设定的规则，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，运行我自己定义的某些“动作”，如：运行某个我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。

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，全部是你自己采集、上传、下载的，任何数据的含义、使用，也由你自己定义。YeeLink 平台只把它们看出是单纯的“数据”，所以，它只是一个数据承载平台。

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，还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 App，还是不能真正打开热水器、启动空调的，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，这就需要借助家用电器“智能化”来做的事情，YeeLink 也是实现不了的。

社交网络融合：在 Yeelink 上存储的数据，可以简单的被手机 API 取回。在手机上编写一个应用，读取 YeeLink 上的数据，并嵌入到你的个人博客上，或者根据规则自动转发到你指定的微博上，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，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、比如 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。（再继续发展，是否会出现面向对象编程中的“类”）？

当你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。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，将在下节：树莓派与物联网服务器通信中介绍。

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，比如一组数据形式为：

{

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 作为你的传感器数据服务器，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/），点击快速开始，按照步骤注册账号。

2）添加一个设备和传感器：

YeeLink 把你的上传数据划分为“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，可以点击“设备管理”，看见自己新添加的设备和设备下的传感器，如图 5-3。

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 5-3 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，得到不断地完善和扩展。目前在 WWW 中使用的是 HTTP/1.0 的第六版。

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：

GET　请求获取 Request-URI 所标识的资源；

POST　在 Request-URI 所标识的资源后附加新的数据；

HEAD　请求获取由 Request-URI 所标识的资源的响应消息报头；

PUT　请求服务器存储一个资源，并用 Request-URI 作为其标识；

DELETE　请求服务器删除 Request-URI 所标识的资源；

TRACE　请求服务器回送收到的请求信息，主要用于测试或诊断；

CONNECT 保留将来使用；

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：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受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。

3）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，分别是：状态行、消息报头、响应正文。

4）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。

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，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。

1）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：

```
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：

```
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：当你访问 r.text 之时，会使用其响应的文本编码进行解码，并且你可以修改其编码让 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，会自动为你解码 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 JSOK 库

1）JSON 数据结构：

JSON 建构于两种结构：

（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 2 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），纪录（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]}]
```

通过输出的结果可以看出，JSON 数据通过 encoding 处理之后，跟其原始的 repr()输出结果非常相似，但是有些数据类型进行了改变。例如上例中的元组则转换为了列表。

在 json 的 encoding 过程中，会发生自动将 python 数据的原数据类型，转变为 json 的数据类型的转化过程，具体的转化关系请参考相关资料。

在上例中，json.dumps()方法返回了一个 str 对象 encodedjson，接下来在对 encodedjson 进行 decode，得到原始数据，需要使用 json.loads()函数（类似 json.dumps()函数的反函数）：

```
decodejson = json.loads(encodedjson)
print type(decodejson)
print decodejson[4]['key1']
print decodejson
```

输出：

```
<type 'list'>
[1, 2, 3]
[[1, 2, 3], 123, 123.123, u'abc', {u'key2': [4, 5, 6], u'key1': [1, 2, 3]}]
```

loads 方法返回了原始的对象，但是仍然发生了一些数据类型的转化。比如，上例中‘abc’转化为了 unicode 类型。由此可见，在 encoding 与 decoding 过程中，并不是一一对应的。

JSON 的 json.dumps 方法还可以实现排序、格式化等功能，不再一一介绍。

5.1.7 上传数据到 YeeLink

有了上述基本的知识，再来看如何将温度数据上传到 YeeLink。以下是将与树莓派相连的 arduino 上的 DHT11 温度传感器的数据，上传到 YeeLink 上的代码。

1）程序代码：

```
import serial,subprocess //导入串口通信库serial
```

```
import time //导入时间库time
import requests,json //导入requests、json库
ser = serial.Serial('/dev/ttyACM0',9600,timeout=1) //设置连接Arduino的串口
ser.open() //打开串口
try:
    while 1:
      res=ser.readline() //读串口数据
apiurl='http//api.yeelink.net/v1.0/device/340581/sensor/377263/datapoints' //地址
apiheaders={'U-ApiKey':'f7e47c1b609e1a952066748fb01d99b1','content-type':'application/json'}
            res0=int(res)
            res1={'value':res0}
              r = requests.post(apiurl,headers=apiheaders,data=json.dumps(res1))
       print res
       time.sleep(.9) //延时
     except KeyboardInterrupt: //键盘中断，停止程序运行
       ser.close() //关闭串口
```

2）代码说明：

上述代码在串口设置和读取方面，沿用上一节课的代码，当时从串口读入的数据，是送到Qt4编写的用户界面的模拟数码管处理函数，进行显示的。

现在与上次处理的不同，是要把数据，送到YeeLink上。

在温度上传短短几行代码中，使用了如下几个关键技术：

（1）YeeLink 的 API URL：在 YeeLink 添加设备和传感器的时候，记下了设备和传感器的编号，这就是此处“apiurl=”后面的一串代码，其中的关键是设备编号“340581”和传感器编号“377263”，这是前述创建设备和传感器时获得的我的设备和传感器编号。

（2）使用了 requests 的 post 请求：语句 r=requests.post(.....)为 POST 请求。POST 请求是向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。

（3）在 POST 请求中，填写了两个“报头”参数，一个是 YeeLink 的用户密码，一个是正文格式说明。'U-ApiKey':'f7e47c1b609e1a952066748fb01d99b1'参数的含义是：YeeLink 要求的该用户的 ApiKey，相当于用户的操作密码。在 YeeLink 的账户|我的账户设置选项下，可以看到你的 API KEY。

图 5-3 查询 API KEY

'content-type':'application/json 参数的含义是指明发送给接收者的实体正文的媒体类型

为 application/json。

（4）上传数据采用 JSON 格式，并做 encoding 处理。

只有几行代码，包含的信息内容还是不少的。

5.1.8 YeeLink 服务器的客户端 APP

YeeLink 提供 iOS 和 android 的客户端 APP 下载。

1）下载 Yeelink 的客户端：

在 Yeelink 的主页上选择“开发者|移动客户端”，选择符合你的手机 OS 的客户端 APP，下载后，完成在手机上的软件安装。

安装完成后，点击 APP，出现如图 5-4 的用户界面：

图 5-4 YeeLink 的客户端 APP 用户界面

2）运行 APP：

安装完成后，打开 APP 图标，点击“我的 Yeelink”，输入用户名和密码后，出现你在 YeeLink 上设备的列表。选择点击你要查询的设备，再选择要看的传感器（图 5-4 左），就出现了如图 5-4 右的数据显示界面。

至此，从 DHT11 传感器上获得的、你周围的温度信息，就通过 YeeLink，传送到了你的手机上，本节、包括前两节课程的系统开发和集成，帮助这个传感器信息，完成了从 DHT11、arduino、树莓派、YeeLink 服务器，到你的手机 APP 的“旅行”。

5.2 开发自己的云服务器应用

如果仔细一点的话，你就会发现，在手机上看到的传感器的实时温度曲线，变化速度是非常慢的。其实，如果换一种传感器，例如：使用声音、亮度、人体感应传感器的话，这种“慢”（实时性）的感觉将更加明显。在现实生活中，如果用这样的系统，做安全监控的话，小偷把你们家都搬走了，传感器的信号可能才刚刚送到你的服务器上。

问题在哪里？整个系统的“实时性”很差。“慢”在哪个环节上了？

5.2.1 搭建苏宁云服务器环境

现在，云计算是一个热门话题，有关的概念，请参考相关资料，这里不累述。

作者选择苏宁云是因为它相对比较便宜，第一次申请，可以得到 200 元的免费试用。200

元可以一个比较长的时间，足够支撑一次实训课程阶段，同学们上网的需要。

1）申请苏宁公有云：

在苏宁公有云界面上，选择“控制台|其他|免费试用”，按照菜单提示，完成免费试用申请，得到 200 元试用费。申请界面如图 5-5：

图 5-5 苏宁云服务器免费试用申请界面

在 1-2 个工作日后，申请或许得到批准，可获得约 200 元试用费，试用费用实时计算，没有使用期限。

获得申请批准后，可以登录到自己的云服务器上，继续申请可用的资源，如图 5-6。

图 5-6 苏宁云的免费使用资源一览

由于费用有限，根据开发需要，上述可用资源中，除了免费的资源外，收费资源中，只选择云服务器（1 个）、公网 IP（1 个）。

2）配置云服务器：

根据苏宁云服务商的操作提示，完成一个最低配置的云服务器（最少费用）的配置。

（1）确定服务器的基本参数：

在新建服务器界面上（如图 5-6），选择服务器的 CPU 个数（1 个，苏宁云按 CPU 个数收费）；内存（1G）、OS 类型（目前有 5 种 Linux 操作系统，不支持 windows 系统）、密码方式（后面介绍）、服务器台数（1 台）、以及服务器名称等。

图 5-6 选择新建服务器的配置

（2）申请一个公网 IP：

公网 IP 是必须要有的，否则，应用建立好以后，还是无法登录云服务器访问自己的应用服务器。这个 IP 是自动与自己的服务器绑定的，一旦申请，请记住 IP 地址（如图 5-7）。

图 5-7 申请公网 IP

（3）确定用户名和密码：

苏宁云服务器有一般密码和 SSH 密码之分，一般密码在用远程终端登录 Linux 系统时，需输入用户名和密码，而 SSH 密码则需要在远程终端连接时，指明采用 SSH 密码方式。请注意两者的区别，二者不可同时使用。

在服务器关机状态下，选择云服务器下自己的服务器，选择“重置”，可以设置或修改服务器密码，界面如图 5-8：

图 5-8 修改服务器密码方式

配置完成后，在苏宁云上，可以看见配置的情况如图 5-9：

图 5-9 苏宁云资源使用情况界面

3）启动服务器：

选择云服务器下自己的服务器（可以有多个，玩大数据时用）。点击“启动”。稍等片刻，服务器被启动起来了。“启动”按钮变灰，而“关机”按钮变蓝，说明服务器已经完成启动（图 5-9）。服务器启动后，可以看到服务器状态变为：“运行中”，以及连接的公网 IP，地址是：218.2.204.233。

图 5-9 启动后的云服务器状态

在图 5-9 界面下，点击云服务器 ID，可以看到这台服务器的更多情况（图 5-10）：

图 5-10 苏宁云服务器的具体运行和配置情况

4）登录云服务器：

云服务器是一台真正的远程服务器。所以，只能提供远程访问的方式登录服务器。在第三章树莓派应用开发中，已经介绍了远程登录树莓派的方法，这里是一样的。

（1）在 windows 下使用 windows 自带的远程桌面登录云服务器：

使用任何一种远程登录软件，登录到云到服务器上，缺省用户名：root，使用刚刚自己定义的密码，或者使用 SSH 密码，但要使用支持 SSH 通信方式的远程终端软件。

用 root 名登录后的当前目录是：/root，可以看到操作系统的版本是 CentOS6.6（或其他选择的版本），服务器主机用户是申请时分配的名字。此时，就可以像使用本地机器一样，使用这台云服务器了。目前苏宁云不提供 windows 系统服务器。

（2）在 windows 下，也可以使用支持 SSH 的 SecureCRT 远程仿真软件登录到服务器上。运行 SecureCRT，输入服务器 IP、用户名和密码。结果如图 5-11：

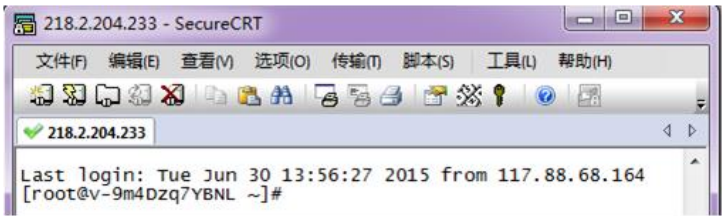

图 5-11 用 SecureCRT 登录云服务器

（3）在树莓派上登录云服务器：

直接在树莓派的 LX 终端上，执行以下命令，也可以登录到云服务器：

$ ssh 218.2.204.233

登录后的结果见图 5-12：

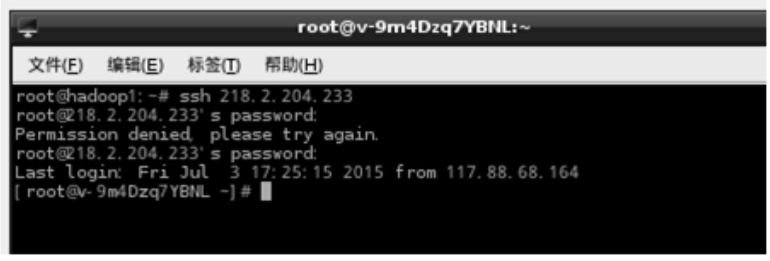

图 5-12 在树莓派的 LX 终端下直接登录云服务器

有一点要注意：在树莓派上，如果我当前的用户名是 root，登录云服务器也就是用 root 名，对方问的密码也是 root，链接成功！但有时你当前的用户名可能是 pi，而云服务器上是没有名叫“pi”的用户的，结果当然是用户名/密码错！

如果是 pi 用户，又不想改为 root，登录的方法是：ssh –l xxxxx 218.2.204.233 其中，xxxxx 是云服务器上已经有的用户，例如：root。

5.2.2 把应用搬到云服务器上去

如前所述，云服务器就是一个干干净净的 Linux 系统，要想搭建什么应用，完全看你自己的需要。作为一种体验和尝试，本小节先把一个原来放在自己电脑上的、现成的 Web 应用，搬到云上去看看。

1）为服务器安装 tomcat6 环境：

Tomcat6 环境的安装，与是否是云服务器，还是自己的电脑，是没有什么关系的，是一样的。这里简单罗列一下安装过程：

由于在云服务器上选的是 CentOS 的 Linux 版本。CentOS 的安装命令 yum 类似于树莓派上的 apt-get，是 linux 系统的自动安装命令，yum install 仅安装指定的软件。

yum （全称为 Yellow dog Updater, Modified）能够从指定的服务器自动下载 gz 包并且自动安装，可以自动处理依赖性关系，并且一次安装所有依赖的软体包，无须繁琐地一次次下载、安装。yum 提供了查找、安装、删除某一个、一组甚至全部软件包的命令，而且命令简洁而又好记。

yum 的命令形式一般是如下：yum [options] [command] [package ...]，其中的[options]是可选的。选项包括-h（帮助），-y（当安装过程提示选择全部为"yes"），-q（不显示安装的过程）等等。[command]为所要进行的操作，[package ...]是操作的对象。

为云服务器安装 Tomcat6 的安装过程是：

（1）通过 yum 自动安装 tomcat6 和 dependences

```
# yum install tomcat6 //安装 tomcat6
# service tomcat6 start //启动 Tomcat6
```

如果访问 http://218.2.204.233:8080/访问不了，那大多是防火墙已经用了 8080 端口，解决方法如下：

```
iptables -A INPUT -p tcp --dport 8080 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 8080 -j ACCEPT
```

如果安装正确的话，就可以在任何的浏览器中看到 tomcat 的默认页了。

2）访问 Tomcat6：

在 IE 浏览器（不是远程终端）上输入服务器 IP 地址和 Tomcat6 的 8080 端口，就可以直接访问 Tomcat6。

3） 配置 tomcat6（如果不打算用 tomcat6 的 manager，可不用此修改，直接跳过去）：

配置文件：/etc/tomcat6/tomcat6.conf + /etc/sysconfig/tomcat6

Tomcat6 home 目录: /usr/share/tomcat6

配置 tomcat6 为 admin 和 manager 用户：

修改文件/usr/share/tomcat6/conf/tomcat-users.xml

```
<!-- The host manager webapp is restricted to users with role "admin" -->
<!--<user name="tomcat" password="password" roles="admin" />-->
<!-- The manager webapp is restricted to users with role "manager" -->
<user name="tomcat" password="password" roles="manager,admin" />
</tomcat-users>
```

此时可以访问 http:// 218.2.204.233:8080/manager/html 和 http:// 218.2.204.233:8080/host-manager/html 来管理 tomcat6 server 和 host。

4）部署 Tomcat6：

方法 1：【使用控制台部署】

访问 Http：//218.2.204.233:8080，并通过 Tomcat Manager 登录，进入部署界面即可。此方法需要前述修改 tomcat manager 的用户名及密码。

方法 2：【利用 Tomcat6 自动部署】

将应用程序复制到 Tomcat6 的 webapps 路径下，Tomcat 启动时将自动加载。

Tomcat6 的 webapps 路径在哪里？根据 Tomcat6 第一页的提示，应该在$CATALINA_HOME 指定的目录下。用 echo $CATALINE_HOME 看一下环境变量的值是什么，显示环境变量 CATALINE_HOME 的值是：

/usr/share/tomcat6

在/usr/share/tomcat6 目录下再看一下：实际的 webapps 在/var/lib/tomcat6 目录下。终于找到了。

在/var/lib/tomcat6 目录下再看一下：已经有几个应用实例了。验证一下 examples 的例子是否可以运行。在 IE 上直接运行这个应用就行啦！结果如图 5-13：

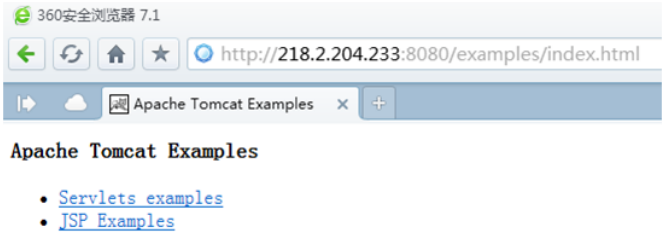

图 5-13 运行 Tomcat6 自带的 Web 应用例子的界面

5）把自己的应用搬到云上：

找到了 webapps 的位置，把在电脑上的 Web 应用，直接搬到这个目录下，Tomcat6 启动时将自动加载这个应用。这就可真的向全世界，发布自己的应用了（难道不需要 ICP 报批备案？注意不要乱来哟）。

操作过程是：

（1）复制文件：通常使用 scp 命令（涉及 SSH）复制文件，另一个简单的工具是 lrzsz。使用 SecureCRT 的好处是复制文件非常简单，而且还是图形用户界面。有关使用 SecureCRT 进行文件传输的方法，在第三章已经介绍，不再重复。

如果需要把一个项目放到 webapps 时，可以将该项目文件压缩为一个 zip 包，用 SecureCRT，上传到云服务器的当前用户的当前目录下。

在当前目录下，执行：

mv ****.zip /var/lib/tomcat6/webapps

为该项目创建一个目录：

mkdir **** （假定为 file123）

把该压缩文件移动到新目录下：

在新目录下，使用解压缩命令：

unzip ****.zip

现在需要重启 tomcat。简单的方法就是重启服务器。

在 IE 上看看是否能看到这个项目？

注意：项目地址：http：//218.2.204.233:8080/file123/zhuce.html。其中 IP、8080 端口不

要解释，file123 是 webapps 下为这个项目创建的目录名，而 zhuce.html 则是这个 Web 项目的入口 html 文件（通常也用 index.html）。在 IE 上，就可以看见你的首页了，如图 5-14：

图 5-14 发布在云服务器上的 Web 应用首页

5.2.3 开发自己的 Web 应用

有了上面的“发布”，似乎勾起了更大的“野心”，何不做一个私人订制的物联网应用网站？

1）我的物联网网站的需求：

本着先简单，后复杂、逐步深入的原则，我私人订制的网络，可以把 DHT11 传感器的温度，显示在手机 APP 上——暂时还是用手机 IE 浏览器吧，iOS 或 Android 的 APP 开发？稍后。同时，还要能在手机上反向控制树莓派的 LED 灯。

需求：

1、在树莓派上，将传感器 DHT11 采集到的温度数据上传到苏宁云 Web 服务器上；

2、在 Web 服务器端，接收上传的数据；

3、在 Web 服务器 HTML 网页中，实现将上传的温度数据，用图标曲线的形式画出来；

4、在 Web 服务器上，设置反向控制按钮，向树莓派发送控制命令；

5、树莓派接收来自 Web 服务器的命令，实现对 LED 灯的控制。

5.2.4 树莓派与 Web 服务器交互

在上一节课已经讨论了树莓派与 YeeLink 服务器的 HTTP 通信的问题，在本节，与上节不同的地方，只是 Web 服务器不是 YeeLink，而是我自己的云服务器了。云服务器只是在“云”上，本质上就是一台 Linux 服务器，所以，接收数据、做出响应的事，要靠我自己了。

Web Socket 的方式有很多种，目前比较流行、简单的传输协议和数据格式是：

发送方：

```
HttpClient client = new HttpClient();
PostMethod method = new PostMethod(URL);//具体 method 里面还可以设置一下编码，header 之类的
//1. 第一种方式，基于 Content-Type='multipart/form-data'形式的表单
Part[] parts = ...;//FilePart 和 StringPart 都可以放进去
```

```
method.setRequestEntity(new MultipartRequestEntity(parts, method.getParams()));
//2. 第二种方式，普通表单
NameValuePair[] pairs = ...;//纯参数了，键值对
method.addParameters(pairs);
client.executeMethod(method);
```
接收方：
```
private void parseRequest(HttpServletRequest request) throws Exception {
  boolean isMultipart = ServletFileUpload.isMultipartContent(request);
  if (isMultipart) {
    DiskFileItemFactory factory = new DiskFileItemFactory();
    ServletFileUpload upload = new ServletFileUpload(factory);
    List items = upload.parseRequest(request);
    for (int i = 0; i < items.size(); i++) {
      FileItem item = (FileItem) items.get(i);
      if (!item.isFormField()) {
        //文件数据
      } else {
        //普通表单数据
      }
    }
```
接收方：
```
  } else {
    Enumeration en = request.getParameterNames();
    while (en.hasMoreElements()) {
      String paramName = (String) en.nextElement();
      String paramValue = request.getParameter(paramName);
    }
  }
}
```

5.2.5 搭建一个 Web 网站

搭建一个 Web 网站的工作，现在实际并不是十分困难，用 DW CS6 软件，可以可视化地、非常简便地、不用写一行代码地完成一个具有相当复杂度的网站开发与部署发布。目前的建站工具和方法也很多，但本质不外是 HTML5+CSS+JS 而已。

如果说 DW CS6 是一种基于建站素材（文字、图片、格式），使用 HTML 的表、层、框架、CSS 格式定义来搭建网站的工具，另一种更简便的方法是采用现成的“框架”。框架的作用，就是把成熟的应用，抽象、提取出来，形成可直接所以的“基础构件”、甚至是“基础结构”，而使用只要在此基础结构之上、用基础构件进行“拼装”，就可以了。这就是框架的作用。软件架构设计在高级阶段，讨论的就是“行业应用框架”。而 SSH 架构也是一种常用的框架。有关这方面的应用，将在下一章进行更深入的介绍。

1）网站开发的 Flask 框架：

Flask 是一个轻量级的 Web 应用框架，使用 Python 编写。基于 WerkzeugWSGI 工具箱和 Jinja2 模板引擎。Flask 也被称为 “微框架（microframework）”，因为它使用简单的核心，用 extension 增加其他功能。Flask 没有默认使用的数据库、窗体验证工具。然而，Flask 保

留了扩增的弹性，可以用 Flask-extension 加入这些功能：ORM、窗体验证工具、文件上传、各种开放式身份验证技术。

（1）最小的 Flask 框架程序的例子：hello.py。

hello.py 的代码：

```
from flask import Flask //导入 Flask
app = Flask(__name__) //程序名

@app.route('/') //资源位置
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

（2）安装 flask 并运行 hello.py 程序：

执行命令：

$ sudo pip install flask

$ python hello.py* Running on http://localhost:5000/。

现在浏览 http://127.0.0.1:5000/，你会看到你的“*Hello World !*”问候。

（3）代码说明：

首先导入 Flask 类 。第一个参数是应用模块的名称。如果你使用的是单一的模块（就如本例），第一个参数应该使用 __name__。取决于如果它以单独应用启动或作为模块导入，名称将会不同，'__main__' 对应于实际导入的名称。

接着，创建一个该类的实例。然后传递给它模块或包的名称。这样 Flask 才会知道去哪里寻找模板、静态文件等等。使用装饰器 route() 告诉 Flask 哪个 URL 才能触发我们的函数。定义一个函数，该函数名也是用来给特定函数生成 URLs，并且返回我们想要显示在用户浏览器上的信息。

最后用函数 run() 启动本地服务器来运行应用。if __name__ =='__main__': 确保服务器只会在该脚本被 Python 解释器直接执行的时候才会运行，而不是作为模块导入的时候。

（4）Flask 应用程序框架结构：

一个网站所需要的程序量当然比上述例子要大得多，网站根据不同的功能，可能会分为多个不同的模块，如果把所有功能都写在一个程序文件里，后期将会很难维护。

现在来看一下实现画出温度曲线的网站结构（图 5-15）（开发架构）：

```
[root@v-9m4Dzq7YBNL wulin]# tree
.
├── main.py
├── static
│   ├── chartkick.js
│   ├── css
│   │   ├── main.css
│   │   └── normalize.css
│   ├── highcharts.js
│   ├── img
│   │   └── Cloud.jpg
│   └── jquery.min.js
└── templates
    ├── get-pi.html
    └── index.html

4 directories, 9 files
[root@v-9m4Dzq7YBNL wulin]#
```

图 5-15 云上的开发架构

其中：程序 main.py 管理整个 WEB 应用的运行；static 目录下是 chartick.js、highcharts.js、

jquery.min.js 三个主要的 javascript 代码文件，以及一些 css 和一个背景照片 jpg 文件，它们是采用 chartick 函数库实现的绘图程序；templates 目录下的 get-pi.html 是与树莓派交互的温度数据采集程序，也是网站的入口。Templates 下面放的都是 Flask 的 URL 启动时的引导模板，一般用来处理显示网站的标题、用户登录等内容。

上述开发框架也是 flask 框架的规范结构，就是什么东西应该放什么地方。

2）绘图工具 chartkick：

基于 Web 网站的画图工具很多，一般用法是将具有绘图功能的 js 代码，嵌入到 HTML 中，在这方面，也有工具可以帮助你实现，而不需要自己费劲地写 js。

Chartkick 是一个图表绘制工具，特点是 UI 美观、使用简单，并且支持 IE6 在内的大多数浏览器。chartkick 可以画 javascript 报表，界面比较美观，其支持加载 Google Charts 和 Highcharts 图形库，而且支持集成 Django, Flask/Jinja2 框架，因此，可以和前述的 Flask 框架，很好结合。

3）服务器部分程序说明：

（1）Flask 主程序 main.py：

主程序 main.py 是用 python 写的：

```
from flask import Flask   //导入 Flask
from flask import request //导入 request
from flask import render_template //导入 render_template 模板
app = Flask(__name__) //作为主程序
a = []
@app.route('/') //资源都在当前目录下
def show():
    return render_template('index.html', datalist=a) //按 template 目录（模板）下的网页显示格式显示
@app.route('/', methods=['POST']) //接收 POST 请求传送的数据
def getDate():
     a.append(request.json['value']) //获得数据
def store(data):
    pass
def get(data):
    pass
if __name__ == '__main__':
    app.run(host='218.2.204.233',port=5000) //云服务器 IP 及端口
```

（2）index.html 文件：

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <title>Anx's cloud</title>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link rel="stylesheet" href="/static/css/normalize.css">
    <link rel="stylesheet" href="/static/css/main.css">
  </head>
```

```
  <body>
    <div class="container">
      <header>
        <h1>Data from raspberry PI</h1>
      </header>
      <section>
          <textarea rows="10" cols="50">
            {% for data in datalist %}
              {{data}}
            {% endfor %}
          </textarea>
      </section>
    </div>
  </body>
</html>
```

（3）get-pi.html 程序：

HTML 头部分

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <title>Anx's cloud</title>//html 文件标题
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link rel="stylesheet" href="../static/css/normalize.css">//使用 css 格式定义
    <link rel="stylesheet" href="../static/css/main.css">
      <script src="../static/jquery.min.js"></script>
      <script src="../static/highcharts.js"></script>
      <script src="../static/chartkick.js"></script>
<head>
```

HTML 文件-显示层部分

```
<body>
    <div class="container">
      <header>
        <h1>Data from raspberry PI</h1>//网页标题
      </header>
      <section>
//定义一个层的显示格式
<div id="chart-12" style="height: 300px; text-align: center; color: #999;
                          line-height: 300px; font-size: 14px;
                          font-family: Lucida Grande, Lucida Sans Unicode,
                          Verdana, Arial, Helvetica, sans-serif;">
    Loading...
</div>
```

```
<script>
//<![CDATA[
new Chartkick.ColumnChart(document.getElementById("chart-12"), {0: 12.0, 1: 12.0, 2: 12.0, 3: 12.0, 4: 12.0, 5: 6.0, 6: 12.0, 7: 6.0, 8: 12.0, 9: 12.0, 10: 12.0, 11: 12.0, 12: 12.0, 13: 17.0, 14: 12.0, 15: 17.0, 16: 12.0, 17: 22.0, 18: 22.0, 19: 12.0}, {"library": {}});   //绘制柱线图
//]]>
</script>
        </section>
      </div>
  </body>
</html>
```

4）树莓派程序：

树莓派上的程序代码与发送给 YeeLink 的代码一样，采用 POST 方法，JSON 格式，只是地址不同，且不需要 ApiKey，不再分析。

5）最后效果：

在 IE 上输入：http://218.2.204.233:8080/wulin/templates/get-pi.html，得到如图 5-16 结果。如果树莓派不断上传新数据，则显示会不断更新。

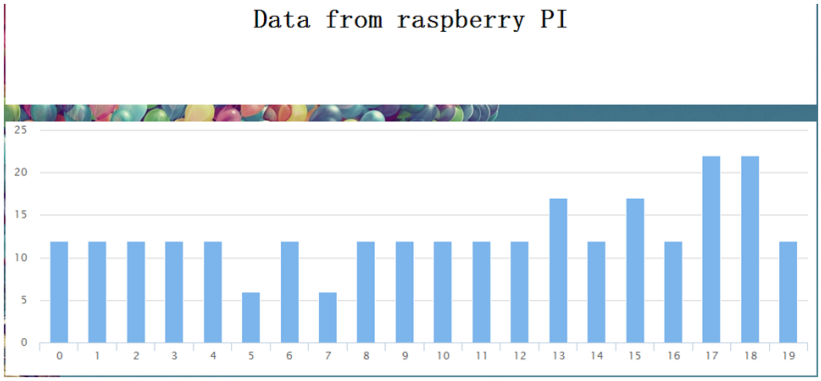

图 5-16 接收树莓派上传的温度数据并用图形显示温度变化

在 IE 输入的 URL 有两个关键内容：（1）苏宁云服务器地址和端口（218.2.204.233:8080）；（2）放在云服务器/var/lib/tomcat6/webapps/wulin/templates/get-pi.html 目录下的 Web 应用项目入口程序（get-pi.html）。

如果想把温度的柱型显示，换成曲线表示，实际上只需要更换一个显示函数，就可以了，同学们自己尝试一下。

5.2.6 在云上直接控制树莓派点亮 LED

由于前端采集、树莓派和云服务器的软件都是自己开发的，因此，在系统开发和集成的灵活性、便利性上，可以满足更多的用户需要。而从项目开发的角度看，之前的所有工作，都可以看成是为了某个目标系统（例如：参赛项目）功能开发的“可行性”实验。已经实现的功能，不是为了满足用户需求，而是为了证明：在技术上，可以实现这些功能——实现技术的可行性上，没有任何问题。这个问题，在下一节“系统开发与集成的可行性分析”中再深入讨论。

1）新需求：

作为例子，用户希望做一个更像 Yeelink 服务器的网站，用户可以通过浏览器或 iOS/Android APP 上访问 Web 服务器，并直接控制树莓派上的 LED 灯。

用户的 IE 显示和实际效果如图 5-17。这个界面很简单，只能表示可以实现反向控制。

图 5-17 在 IE 上控制 LED 灯

使用 flask 进行 python web 开发。

在云服务器上进行一个 web 开发，可以用 Tomcat，当然也可以选其他所谓轻量级的 web 开发框架。Flask 前已介绍过的，用来在 Web 服务器上进行 web 开发的、基于 python 的轻量级 web 开发框架。

1）在服务器上安装 flask 开发环境：

假定服务器系统的 OS 是 debian，其他 Linux 系统的命令可能稍有差异：

ssh 登陆服务器后，执行：

$sudo apt-get install python-pip

$sudo pip install flask

环境就搭好了。

2）测试：

在服务器上写了一个小型的 web blog：emdlog，放在 github 上了，这里可以直接拿来进行测试：

$git clone https://github.com/embbnux/emdlog.git

$cd emdlog/emdlog

$sudo apt-get install sqlite3

$sqlite3 db/flaskr.db schema.sql

$cd ../

$python runserver.py

然后浏览器访问：raspberry_ip:2000，应该就会看到一个很简易的 blog 站。

Runserver.py 的代码：

```
# welcome to my blog : Blog of Embbnux
# url:http://www.embbnux.com/
# author : Embbnux Ji
from rpicloudmanager import app
app.run(host='0.0.0.0',port=2000)
```

前面介绍了在服务器上使用 flask 开发 web 框架，在下面，用树莓派可以很容易的通过 python 来操作 gpio，然后可以通过 web 来控制树莓派的 gpio。用户可以通过 IE 浏览器，访问 web 页面，直接操作 raspberry 的 gpio 底层，或者也可以通过手机 app 发送 post 或者 get 等请求，来控制树莓派的 gpio,这样岂不是很妙！

3）处理 web 请求：

在服务器上使用 flask 进行 web 开发很方便，这里使用 post 来处理 gpio 操作请求：

```
@app.route('/gpio/<int:id>',methods=['POST']) def
gpio_led(id): if request.method == 'POST':
GPIO.setmode(GPIO.BOARD)
```

```
if id<100:
GPIO.setup(id,GPIO.OUT)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(id,GPIO.OUT)
GPIO.output(id,False)
else:
GPIO.setup(id-100,GPIO.OUT)
GPIO.output(id-100,True)
return redirect(url_for('show_index'))
```

5）网页控制按钮的 HTML 文件：

有了 web 请求处理，还需要写一个用来给 IE 用户显示操作按钮 view 的 HTML 文件：

View.html

```
<form action="/gpio/11" method=post>
<input type=submit value="led on" />
</form>
<form action="/gpio/111" method=post>
<input type=submit value="led off"/>
</form>
```

6）运行 web 程序：

web 工程代码如下：

```
$git clone git@github.com:embbnux/RpiCloudManager.git
cd RpiCloudManager sudo python runserver.py
```

代码如下：

```
# welcome to my blog : Blog of Embbnux
# url:http://www.embbnux.com/
# author : Embbnux Ji
from rpicloudmanager import app
app.run(host='0.0.0.0',port=2000)
```

通过浏览器访问 http://your_raspberry_ip:2000 就可以了，效果如图 5-17：

5.3 将本例改造成电梯群控系统

通过前几小节课程，同学们已经完成了一个小型物联网应用系统的开发与集成项目。这个小型物联网应用，在 arduino（传感器）、树莓派和云服务器三个平台上，分别开发了各自的应用模块，并且通过不同的方式（主要指信息传输方式），把三个模块集成在了一起，获得了初步的系统开发体验。

但是，不论上述系统可以集成多少传感器、可以有多少显示样式，它们毕竟只是“闭门造车”的实验，没有真实的应用背景，因此，所获得的感受是有限的。本节尝试将已经完成的系统，“移植”到电梯群控系统的真实背景下，即把这个系统经扩展，成为一个“真实”的电梯群控系统，以此来进一步体验系统开发和集成，在真实应用背景下，是一种什么情况。所以，本节是前一个项目“可行性”研究的实用化。本小节为扩展部分，如果时间不够，可略过不讲。

《软件架构实践教程》第六章 6.3 小节“电梯控制系统概要设计与实现”讨论了电梯控制的各个子系统以及相互的逻辑关系。在那本书里，并没有讨论具体的实现细节。本小节不

妨将其作为案例，继续探讨：如果按其分析和设计的电梯控制系统方案，在具体的实现平台（arduino 传感器、树莓派和云服务器）上，进行实现。

5.3.1 电梯群控的物理架构与系统集成

按照《软件架构实践教程》6.3 小节的概要设计与实现方案，一栋大楼可能有很多部电梯，需要实行“群控”。群控系统在物理上，分为：电梯楼层子系统、轿厢子系统与总控子系统。上述三个子系统分别被部署在每个楼层、每个电梯轿厢内及大楼的电梯控制室内。从需求分析获得的电梯控制 OMT 模型如图 5-18：

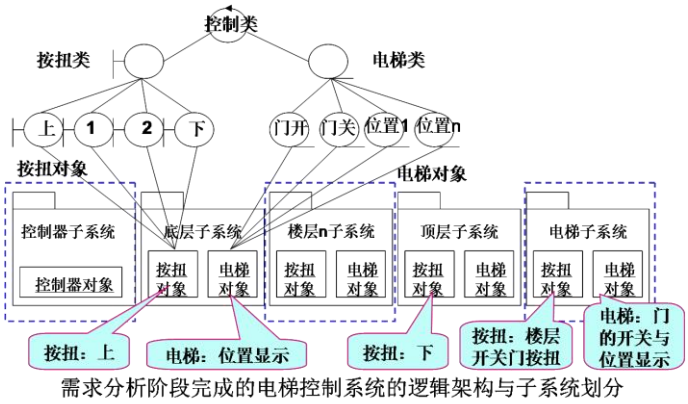

图 5-18 电梯群控系统 OMT 模型

从实现角度看，如果不考虑多部电梯的“群控”需要，显然，Arduino 应该装在每部电梯的每层楼的电梯门上，充当楼层控制子系统的角色；树莓派应该装在每部电梯的“轿厢”里，是轿厢子系统，而云服务器应设置在每栋大楼的电梯控制室内，负责总的控制。

似乎电梯群控系统的物理架构、以及所对应的子系统划分，就是这样“客观”、“直接”、“简单明了”地被划分和定义清楚了。

5.3.2 电梯群控的运行架构与功能边界划分

物理架构确定了之后，各子系统的功能划分是否就是很明显、简单的事情？还有讨论的必要吗？在单部电梯控制模式下，似乎确实已经很清楚了，先简单罗列一下：

（1）arduino：arduino 部分实现用户上下电梯的请求（按钮、显示按钮被按下）的上传。同时，通过电梯门上的按钮灯的点亮和熄灭，告知使用者接收到了上/下电梯的请求。arduino 还需要显示电梯当前的运行状态（当前电梯上/下运行方向显示），这个信息来自树莓派。

（2）树莓派：树莓派被放在每部电梯的“轿厢”内，它接收 arduino 的信息，并上传到云服务器，也接收云服务器的信息，下达给 arduino。它自己还有接收“轿厢”内按钮、显示，以及控制电梯电机运行、控制电梯门开关等功能。

（3）云服务器：是否在云上不是关键，如果需要实现全城的电梯安全监控，那一定是在云上的了。群控服务器实现控制和调度，安全监控等。

5.3.3 电梯群控的逻辑架构与系统信息交互

在《软件架构实践教程》第六章中已经分析，简单按照图 5-18 的 OMT 模型，来实现电梯控制，当只有单独一部电梯的时候，即不考虑多部电梯的“分布、并发、实时”需求的时候，似乎问题不大。但是，如果有这些“关键质量属性”要求，则系统将难于满足。具体分析请见该书的相关部分。实际上，接下来的逻辑架构实现，就自然会涉及到单部电梯控制与多电梯控制的差异与不同。

逻辑架构研究系统的各子系统之间的逻辑关系，实际上是各子系统的连接关系（交互关

系）。而确定交互关系的关键，是各子系统之间的功能边界，就是《软件架构实践教程》6.2.7节所谓“子系统划分的关注点分离”。

只有明确了谁做什么、谁不做什么，才能确定“关注点”在什么地方，如何将它们分离开。关注点分离清晰、逻辑关系才能划分清楚。而“关注点”随着需求的不同而不同。

如果只控制一部电梯，电梯各子系统之间的“交互”是“线性（顺序）”的、单进程的。而当多部电梯同时、分布、并发、同步控制的时候，请求队列、运行状态、调度协调则必须是多电梯一同发生、一并考虑的。从运行逻辑上看，电梯的请求、状态、控制是三个“并发”、相互“交互”、实时响应的进程，电梯控制系统的逻辑架构如图5-19。

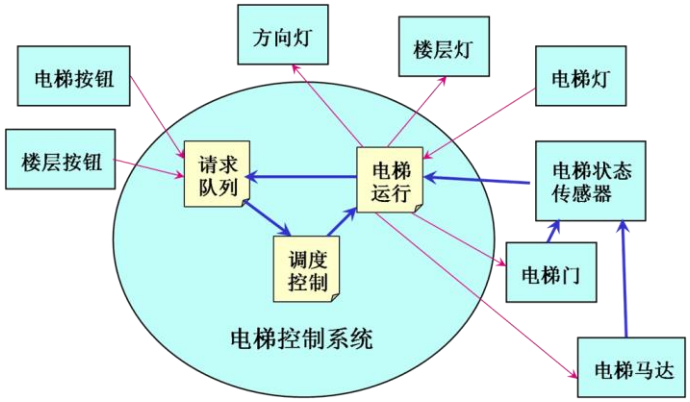

图 5-19 电梯群控系统的逻辑架构

而电梯群控的内部运行逻辑如图5-20所示：

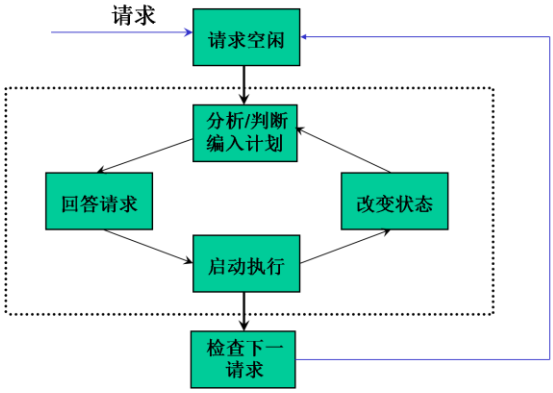

图 5-20 电梯群控系统的运行逻辑

所以，在具体实现上，单部电梯控制模式和多电梯控制模式，arduino 与树莓派之间、树莓派与云服务器之间，甚至 arduino 与云服务器之间，相互传什么信息、哪些信息要上传、哪些可以不上传，而由子系统自己处理，应该是很不同的。

在单部电梯模式下，从 arduino、到树莓派，在到云服务器，可以设计为一条“三点一线”的“责任链”，请求和响应是“一一对应”的。如已经实现的系统，请求信息上传、控制命令下达，系统单线程运行。在这个模式下，各自上传什么信息、下达什么命令，各自担负什么角色是简单、明确的。

多电梯模式下，情况不一样了。

首先，在 arduino 与云服务器之间，是否需要增加直接连接（请求导致调度变化）？否则，只能通过树莓派“二传”。“二传”方案当然会影响系统总体的“响应速度”，也会带来通信过程的复杂性。

其次、arduino、树莓派和云服务器是在“分布、实时、并发”环境下各自运行的，而且 arduino 和树莓派都还不止一个。Arduino 和树莓派对实时要求，能满足多少？三者之间的并发通信，如何同步和协调？arduino 和树莓派在实现并发通信的同时，自身还有很多其他任务，运行速度、响应时间等承受能力够吗？这都是原来“单机”系统没有考虑过的问题。

再有：在多电梯模式下，作为中间层的树莓派，到底扮演一个什么角色？如果三者之间的行动关联设计得非常“紧耦合”，则控制的“实时、并发性”会很好，但模块之间的“紧耦合”会带来系统控制的复杂度大大提高、系统的灵活性和可扩展性降低。反之，效率就会降低。

看来，真的要把设计落实到具体实现，还是有很多问题要讨论的。

作为练习，请同学们以电梯群控为目标，先从具体“实现方案”开始，研究电梯群控系统在 arduino、树莓派、云服务器上的实现方案，并验证实现方案，能够满足多电梯的“分布、并发、实时”控制功能，经得起测试的检验。

5.3.4 电梯群控的可行性分析

在《软件架构设计实践》课程中，学生多在笔记本电脑上实现这个电梯群控系统，只要方案正确得当，实现上基本没有技术问题。即没有上述讨论的、真正的响应速度、并发通信协调等问题。因为电梯控制的三个核心进程之间的交互，都是在同一台笔记本电脑上、采用内部进程通信方式实现的。而在本例中，是在三个真实的、物理、逻辑系统（arduino、树莓派、云服务器）之间进行的，而不是三个“虚拟”的系统。

现实中的这三个系统的差异实在太大了，而正因为如此，所谓“系统集成”，才真正体验到“不同系统”之间“集成”时，必须考虑的系统与系统之间的匹配问题。这样的匹配，不在真实环境下“试验”，是无法知道的，这就是“可行性”分析。

以下就使用 arduino、树莓派、云服务器构成的系统，能否满足电梯控制的“分布、并发、实时性”需求，进行可行性论证。有关电梯控制的“分布、并发、实时性”需求的详细内容以及测试验收标准，请见《软件架构设计实践》6.3.6 节。

1）电梯控制系统实现的“分布性”分析：

在多电梯的控制系统中，“分布性”是物理分布问题，包括多个 arduino 和树莓派被安装在多栋大楼、每个楼层和每个轿厢中。“分布性”的实现，一是多个这样的子系统与云服务器的连接，二是在运行与逻辑处理上，树莓派、云服务器都必须满足“一（树莓派或云服务器）对多（arduino、树莓派）”的关系。

在树莓派和云服务器上运行的系统，都是 Linux 系统，支持多进程/多线程，即使采用“轮循”策略，只要处理速度足够（当然，同时控制的设备数是关键），响应时间是可以保证的。

2）电梯控制系统实现的“实时+并发性”分析：

其实，“分布、并发、实时性”这三个需求是相互关联的，“分布”反映的是被控制“点”是有多个（分布在不同的地理位置，而对于控制系统而言，只要能够接受到信息（不考虑因地理位置造成的信息传输的延时、误码），位置是无意义的。“实时”是“立即”（从需求分析的角度讲，使用“立即”一词描述“实时性”是很不“专业”的，此处为抽象分析），而“并发”是“多点+立即”。

所以，这里把“实时与并发”联系在一起考虑，在实现上，就是同时“实时”响应多点的请求。

Arduino、树莓派、云服务器对于同时响应多点请求的能力如何呢？等同于回答以下问题：

（1）多点的“多”有多少？

（2）实时的“实时”要多“快”？

而这两种是相互关联的，多了就快不了，快了就不能很多。具体要用“试验”的方法，进行测算。所谓可行性分析，就是看具体测算的结果，才能回答：并发的实时，到底可以达到什么程度。

在正式的系统开发之前，编写模拟程序、进行连接测试，获得相关数据，是可行性分析的主要工作。分析不是只在“纸上谈兵”的。

5.4 参加物联网创意大赛

在本章的最后，将简单讨论有关“参赛”的问题。

实训课程与参赛有什么关系？在本教程第一章已经介绍，作者认为：实训课程是软件专业重要的辅助课程，在整体设计上，应在明确的专业培养目标指导下，根据学生实际情况，进行实训课程体系的完整设计和规划，并涵盖学生在校各阶段（年级）。为此，作者建议按三个阶段，进行实训课程设计和组织实施。而“参赛”则是在实训的最后阶段，进一步综合运用课堂教学、实训课程训练的教学成果，检查、验证学生学习成果和教学效果的较好方法。

参赛的创意要求、实战性、综合性等特点，完全可以承担上述期望的目标，有关意义和价值不用累述。不管出于何种目的和动机，参赛对于学生而言，毕竟是有百益而无一害的。

5.4.1 参赛的项目层面考虑

大赛是怎么回事，应该如何参赛并取得好成绩？作者曾多次带队参加各种比赛，也获得过不错的成绩。在作者主编的《基于 VSTS—软件工程实训教程》（清华大学出版社 2011 年 6 月）一书第二章中，专门讨论了有关参赛的一些问题，这些问题可以归结为基于参赛项目层面的考虑。

罗列一下这一章的内容，包括：

- ✓ 认识我们的目标：
  - ■ 微软创新杯——Imagine Cup；
  - ■ 大赛主题与选题；
  - ■ 理解大赛主题；
- ✓ 有一个好的点子：
  - ■ 基于传统问题域的创新选题；
  - ■ 关注新的应用领域；
  - ■ 寻找新的技术应用热点；
  - ■ 参考各类软件创新大赛的获奖选题；
  - ■ 从既往的项目中选择课题；
- ✓ 选题的目标设计与价值评价：
  - ■ 目标与方案；
  - ■ 最初的目标：大赛通知；
  - ■ 第一个交付成果：《项目计划书》；
  - ■ 再谈目标——大赛的评价标准；
- ✓ 课题的难度与可行性：
  - ■ 何谓“难度”；
  - ■ 难度与可行性的关系；
  - ■ 可行性的非技术性考虑；
- ✓ 确定项目的范围：
  - ■ 为目标而确定范围；

- 考虑条件和资源；
- 修剪你的“范围树”；

✓ 项目团队组建与项目初步规划：
- 角色与分工；
- 项目总体规划；
- 为第一次提交，制订更详细的工作计划；

✓ 实训项目案例——《ATM 扩展项目计划书》：
- 参赛作品构思的创意与价值；
- 参赛作品的目标实现形式；
- 参赛作品目标实现的可行性；
- 团队组成与角色分工；
- 项目时间进度表。

由于篇幅所限，不再重复上述内容。这一章的内容从参赛的目标选择（大赛要求的理解）、创意产生的思路和方法、参赛项目目标描述和价值评价、可行性分析方法等面向参赛选择的父母，以及确定项目目标范围、组建团队和项目实施规划等具体参赛项目组织实施方面，进行了探讨。当然该书以后各章继续始终围绕项目开发过程展开，后面还有很多与大赛项目实施过程有关的部分，请有兴趣的同学参考。

仔细看一下上述部分，可以发现，其大多是围绕参赛的项目层面展开的。所谓项目层面，主要是指参赛项目的目标、创意、价值、范围、分工和初步实施规划。这是项目管理的“启动”阶段的主要工作。因此放在《软件工程实训教程》的第二章，所以更侧重软件工程过程、软件项目管理。由于侧重点不同，有关创意与价值、可行性等“技术性”问题，没有、也不可能谈的太深，而这正是本小节下面打算继续深入探讨的问题。

5.4.2 从获奖项目中了解参赛的机会与目标

这里并不将参赛的机会，仅局限于物联网方向。实际上，适合学生参加的创意开发比赛很多，并非只有物联网一个主题。以物联网技术和应用为核心内容的应用项目，都是目前各类大赛的“宠儿”。因为，物联网是热点。

1）认真进行“赛事”选择：

不是什么比赛都可以参加、随便想一个题目就去参赛的。作者曾作为评委，在一些大赛的第一阶段，看到很多这样的“撞大运”的学生项目，当然其命运是可想而知的。为什么有这么多学生，这样“随便”地参赛呢？首先因为“大赛”获奖所带来的“价值”，越来越被学校和用人单位认可。其次是国家、学校在教学改革中，越来越提倡创新和创业，作为这方面的业绩考核驱动，学校也开始重视大赛这种“看得见”、能出“成绩”的工作。这就造成目前大赛名目繁多（甚至有“去大赛网”：http://www.godasai.com/；“全国大学生比赛信息网”：http://bisai.172xiaoyuan.com/等网站），可见“热”的程度。成为“热潮”，难免“鱼龙混杂”。有办了十多年的，如微软的“创新杯”，也有很多新的大赛。这无疑推动了中国学生扎堆（如前几年的奥数、奥信，一旦中国这边降温，总体成绩立即下滑）的情况。

在上述两个“大赛网站”上，对各类比赛也进行了分类。比如：“去大赛网”按国内、国际；专业，而“大学生比赛信息网”则分得更细，如：创业、科技、金融、设计、营销策划、应用开发、广告创意、歌唱、等 28 类。这就有助于同学们分清目标属性，而不是盲目地投项目。在选择了合适的大赛之后，就是仔细研读大赛规则，理解主办方的方向、意图和评价标准。这在上一小节介绍《软件工程实训教程》第二章的第 1 小节：“认识我们的目标”中专门讨论——如何理解大赛主办方的意图。

2）有一个好的“点子”：

每个同学当然都希望自己能有一个“艳惊四座”的参赛项目，好“点子”不是凭空“拍脑袋”想出来的。很多年“高考指挥棒”和社会家庭大环境背景共同作用下，造成我们的同学，脱离社会生活、脱离实际，发现问题、分析问题和解决问题、创新创意的意识和能力，普遍比较差。不能说现在的同学不聪明，缺乏的是上述这些方面的“训练”。本教程希望是一本从架构师的角度，把学生从只知道一点 C++/java 语法、只会写简单的、基于控制台界面的“排序”程序的学生，培养成具有一定计算机逻辑思维能力和系统开发能力的、成熟的软件程序员的教材。同时，也希望能够在这个培养过程中，更多地培养学生的业务能力，而不是一个单纯的“码工”。

如果你不会“过日子”，最简单的学习方法是“看看隔壁邻居一日三餐是怎么过的”。如果你没有什么好的点子，先看看现在“拿奖”的哪些同学，做了些什么。

微软 2015imagine 大赛的几个获奖作品，可以作为一个例子：

（1）荣获“世界公民”类别特等奖得主，来自四川大学的 Geek&Dentist 团队所研发的 eDentist 口腔诊断平台，通过将用户使用手机或外部摄影设备拍摄的口腔图片上传至 Azure 云平台进行数据处理，结合调查问卷分析用户口腔问题，提出针对性建议并为用户推荐附近的口腔医院（图 5-21）。

图 5-21 四川大学的 eDentist 口腔诊断平台

（2）荣获“最佳创新”类别特等奖的东南大学 Geosaurus 团队，研发了一套名为 Matrix，基于智能手机将 Windows 应用拓展到虚拟 3D 世界的普适性解决方案。通过将 Windows PC 的计算能力、手机的便携性和强大传感能力相结合，重构 VR Windows，Matrix 让使用者可以在虚拟 3D 环境中继续 Windows 上的工作，还能以头瞄、磁感、超声波手势和红外追踪等多种人机交互方式与 VR windows 进行（图 5-22）。

图 5-22 东南大学的 3D 虚拟现实

（3）结合增强现实技术研发的可穿戴设备，也成为了本届“创新杯”大赛的热门类别。“世界公民”类别一等奖得主北京邮电大学 ASTROMAN 团队的作品——增强现实头盔 ASTROHAT，正是通过增强现实及传感技术，通过云平台数据，为用户带来沉浸式观星体验（图 5-23）。

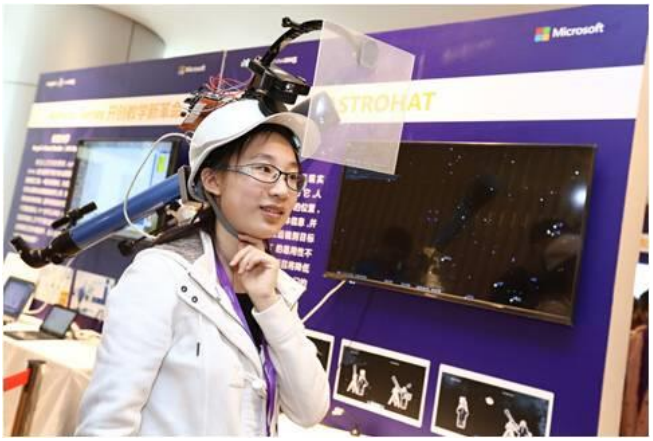

图 5-23 北邮的增强现实头盔 ASTROHAT

3）教练团队：

看别人的项目，不是仅仅看到人家拿了奖。更主要的是看，在这个“方向”上，别人已经走多远，所走过的“技术路线”是什么，在这个方向上，对我有什么“借鉴意义”。

这就像做科研课题、写论文，要想论文能够发表，必须选择一条别人可能已经走过的路线、但是别人没有达到的目标高度。当然，能“另辟蹊径”更好，但那要难得多。如果两者都不是，则你的科研成果是豪无意义的。参赛虽然没有写论文那么高的要求，但是，思维方式是一样的。下一小节专门讨论对获奖项目的技术分析。

与写论文类似的是：某一领域的科研课题，不是今天要发论文了，今天来找课题，而是需要在这个领域长期的、数十年、甚至是一辈子的钻研，才能够获得一点点突破。对学生来说，如果没有这样深厚的基础，就需要“导师”，在方向上给予指点和把握。参赛课题也是一样，因为大赛越来越多，热度越来越高，当然水平和要求会水涨船高。这就不是一个学生所能承担的了。很多学校组织了“教练团队”、一代一代地带领学生参加比赛，自然能够在众多参赛团队中脱颖而出，取得好成绩，这也是知识和经验的积累。

带着自己的目标，看别人的项目，主要看什么？以下主要从三个方面，讨论关注点。

5.4.3 关注数据来源

数据来源是一切计算机应用系统的源头，而更是物联网应用的根本。好的创意，首先是从数据源头开始，然后才是结合数据采集、传输、处理，达到数据应用层面。

以微软 2015imagine 大赛的几个获奖作品为例：四川大学的 Geek&Dentist 团队所研发的 eDentist 口腔诊断平台，通过将用户使用手机或外部摄影设备拍摄的口腔图片上传至 Azure 云平台进行数据处理。数据来源是被检查者的口腔图像，用这些图像数据，经分析处理，构成口腔普查的基础。东南大学 Geosaurus 团队，研发了一套名为 Matrix，基于智能手机将 Windows 应用拓展到虚拟 3D 世界的普适性解决方案。数据来源是通过手机的传感能力，还能以头瞄、磁感、超声波手势和红外追踪等多种人机交互方式，实现与 VR windows 进行操作。而北京邮电大学 ASTROMAN 团队的增强现实头盔 ASTROHAT，正是通过增强现实及传感技术，通过云平台数据，为用户带来沉浸式的观星体验。

三个应用，几乎都可以归为物联网应用。三个应用的原始数据，一个是图像、一个是通过手机获得的操作数据，另一个是通过安装在头盔上的传感器，获得人的头部的位置、动作

等数据，是变相的操作（头部运动）数据。归纳起来看，前一个是操作者获得周围的（口腔）的信息，后两个是虚拟现实信息。所以，数据来源不外部操作者的外部世界和操作者本身的自身的世界。

5.4.4 关注数据采集方式

三个应用的数据采集方式，第一个是摄像头，第二、三个是重力传感器、以及其他辅助的定位、角度、距离等传感信息采集。从这些方面看，并没有什么更复杂的技术。而定位精度、旋转角度、微距变化与识别等，可能是其中比较难处理的技术关键。

5.4.5 关注数据应用

原始数据的应用模式，才是这三个项目的关键。特别是四川大学的口腔诊断系统，凭借得到的大量被检查者的口腔图片，进行图像处理、症状特征识别，然后在进行统计分析。口腔内的图像处理和识别，是否比人脸识别更复杂？所以，其应用的难点在这里。东南大学的虚拟 windows 操作，在获得用手机重力替代鼠标键盘的操作信息后，如何操作 windows，由于没有看见实际操作，不知道到底与一般 windows 操作，有什么不同。同样，北邮的项目，在观星体验中，添加了根据头部运动的操作信息，给观星本身，带来什么不同，也不得而知。但可以想象，不外乎可能省去一些人工操作（类似残疾人使用设备）等等。

在做了上述三个方面的简单分析之后，可能会得出这样的结论：就是获一等奖的项目，也不过如此嘛。当然，我们不知道很多细节，什么事情看上去简单，真要做起来，就不是那么容易的了。

分析三个项目的例子，至少帮我们开拓考虑问题的思路：数据从哪里来？到哪里去？用这些数据做什么事？达到什么目的、意义是什么？

5.5 阶段课程小结

什么是系统？什么是系统开发？什么是系统集成？从架构师的角度看，如果把要集成起来的各个系统（如 arduino 系统、树莓派系统、云服务器系统等）看成是最终应用系统的“子系统”，系统开发就是各子系统（模块）的开发、系统集成就是把子系统（模块）集成为系统。

所以，系统开发与系统集成的第一个区别，是系统集成首先要考虑系统如何被划分、然后，如何被集成。（当然被集成的“系统”本身又被划分为多个子系统，系统与子系统都是相对的。）而系统划分则是架构师进行系统概要设计的第一件工作。

与一个“单机”应用系统的功能划分不同的是，在前几节课程中完成的应用系统开发过程中，各子系统具有相对独立的硬件平台、系统环境，运行各自独立的应用系统程序，各自构成了一个相对独立的“物理系统”。所以，有时讲系统集成，更多的是把不同物理平台上的系统集成为一个完整的系统。

与“单机”应用系统的功能划分相比，基于不同物理平台的系统功能划分，相对比较容易。由于整个系统的功能、运行、逻辑架构、被鲜明的分解到不同的物理架构的子系统中，因此，各物理系统的功能、甚至包括连接和连接关系，都是“客观”的、“使命使然”的。例如：上两章和本章共同完成的系统中，arduino、树莓派和云服务器的功能使命是自然的、一般无法相互替代。

除了本例的基于物理和网络的子系统划分方法之外，子系统划分还与其他机制有关，例如：基于责任层次的、基于状态转换的、基于其他如：开发架构、接口规范约束、以及非技术因素的划分方法，具体请见《软件架构实践教程》第六章。

一名合格的架构师在设计系统架构的时候，应该站在整个系统全局的高度，在“物理”、“运行”、“逻辑”、“开发”、“数据”等架构视角中，首先考虑的是“物理”架构的需要。因为物理架构是基于客观的系统物理布局形成的，最先被确定下来的系统架构需求。例如：系统是“单机”版还是网络版，然后才是网络版的 C/S 架构还是 B/S 架构，然后在客户端和服务器之间，是采用 socket 方式，还是 HTTP 方式进行通信等。不同的应用系统会选择不同的软件架构，不但原因很多，而且会在不同的架构方案中取舍、平衡、折中，甚至综合，这就是架构师的工作。在我们已经完成的项目中，系统架构是非常简单、明确、“客观”形成的，这只是一个特例而已。

显然，之前完成的“实验”系统非常简单，而其他应用系统，可能就不是这样。在不同的应用中，树莓派和云服务器的功能划分，就有再讨论的必要。例如：树莓派仅仅是一个数据的汇总和上传/下达的平台吗？自身不需要做任何数据处理？显然，在一个应用中，分层、分级的数据处理和控制，将更有效率。有些数据在树莓派这一级，就可以进行处理，而不需要上传。当达到某个控制阀值的时候，经授权，树莓派这级，就可以进行简单的报警、信息传递与提示、甚至反向控制等。这完全看监控系统的管理模式而定，而不是由计算机系统本身的构成决定。不同的功能划分，就导致不同的子系统之间的交互方式的不同，这是显然的。