node.js 成为新型物联网开发的催化剂

1、什么是 node.js

提到 node.js，就不得不先提一下 javascript（以下简称 js）。我们不要被 js 的外表所迷惑，其实 js 和 java 一丁点关系都没有。js 是一种最基础的网络脚本语言，用于 web 应用开发，通常是被嵌入在 HTML 中以实现自身的功能。所以 js 只能在浏览器中运行。但 2009 年出现的 node.js 彻底改变了这一局面，使得 js 可以脱离浏览器，在 node.js 引擎中运行，似乎这还不足以激动人心，但 node.js 一重磅功能是使 js 运行于服务端！这无疑对前端开发工程师们有里程碑的意义，终于可以平滑的接触 node.js 后端开发了。这些似乎看起来和嵌入式八竿子打不着。但由于 node.js 引擎跨平台特性，理所当然的在嵌入式平台上运行起来，并且提供了事件驱动，非阻塞 I/O 模型，轻量，高效，跨平台等优秀特性，适应了新型物联网嵌入式开发的潮流。

2、物联网嵌入式开发现状和困境

我们知道，物联网嵌入式开发都是以具体的求功能为出发点，在某个具体平台上实现功能代码，开发上特点也很鲜明：

a、入门门槛较高，知识点非常零散。

b、平台依赖很强，换平台比较痛苦。

c、开发中出现的问题较难定位，往往涉及到多个技术层面协同工作。

d、公司多选择闭源，对整体软硬件生态的贡献接近于零，阻碍了开源的发展。

e、实践多于理论，此领域的大牛多是不断的编程、验证、思考等一步一步成长起来的，几乎没有捷径。

嵌入式开发的这些特点就注定了它的神秘、难度大、团队作战。然而，开发者们总是不会拒绝更简单的语言、更友好的封装、更易用的平台的。如何更快的将创意变为现实？在国外，以 Raspberry Pi 为代表的微型卡片电脑使得神秘的嵌入式平台更加平易近人，它在卡片大小的体积上提供了完整的电脑的功能，可以开发多种应用程序和丰富功能的硬件产品，正由于其高度的软硬件整合使得大大降低了开发门槛，让创业者可以轻松的将创意转化为现实，甚至儿童都可以在其上边实现自己的想法。还有以 Arduino 为代表的入门平台使得微控制器开发变得极为简单，比如拿读取温度传感器的数据来说，在普通单片机上实现可能需要半天时间，而在

Arduino 上可能只需 10 分钟。对于小白来讲，使用 Arduino 大大降低了开发难度。反观国内这一领域起步较晚，有个别嵌入式板卡供应商在陆续转向这个方向。例如 friendlyarm 出品的 nanoPI 系列开源硬件、Firefly 出品的四核卡片电脑开源平台 Fireprime。随着国内开源软硬件的增多，国内对整个开源生态的贡献逐步加大，开源的思路和跨平台的方式逐步被越来越多的开发者青睐，我认为这和传统嵌入式开发已经有足够的区别，暂且称为“新型物联网开发”，归纳出了以下几个特点：

a. 由“支持网络”上升到“基于网络”。

b. 软硬件高度整合，将复杂的底层处理隐藏起来，留给用户友好的 API。

c. 扩展变得极其简单，就像电脑上安装软件一样。

d. 设备互联更重要，甚至可随处推送并部署代码，随处访问。

e. 一般会有社区、github、wiki 等伴随成长。

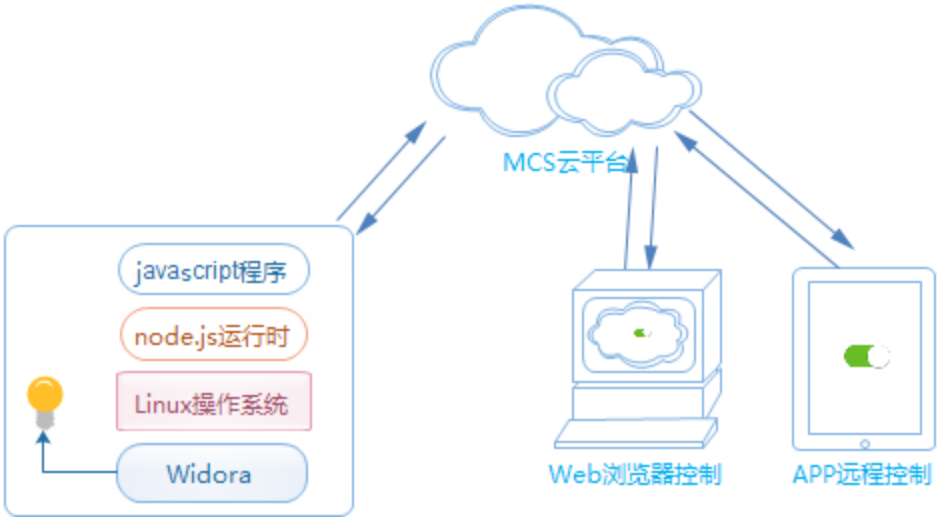
3、基于 node.js 的新型物联网开发框架

目前流行的新型物联网框架有 iot.js、Duktape、tessel、Three.js、icoolpy、johnny-five、linkit 等等，且绝大部分新型物联网开发框架不约而同的选择了 node.js 平台为核心。Widora 就是这样一个初生的极简开源硬件，基于 MT7688 MIPS CPU 运行 OpenWrt 系统，也正在融入 javascript 带来的基于 WEB 的新型物联网生态。

js 俨然成为 WEB 世界里的“二进制”，另外 Widora 提供了 MT7688 以及 OpenWrt Chaos Calmer 操作系统，等同于打包了底层软硬件。这中间缺了一道“桥梁”就是 node.js 运行时，在 Widora 联网的情况下，只需要一条“#opkg install node”命令安装 node.js 和 npm 让这个桥梁架设起来。

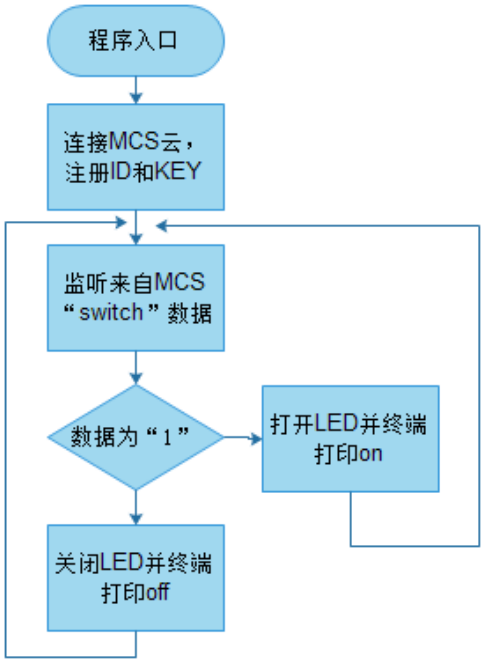
4、基于 node.js 新型开发方式案例：云端控制 LED 实例

我们通过联发科 MCS 云平台和 Widora 开源硬件创建一个可以云端远程控制的 LED，来展示如何通过 node.js 实现远程网络数据控制功能，以及如何用这种全新的开发方式实现。大致的架构如下：



在 MCS 云端建立一个“widora”原型，并添加一个叫“switch1”的测试装置，新增资料通道类型选择为“开关”，创建完会得到 DeviceId 和 DeviceKey，这两个标识很重要，本地的应用程序会用到。此时应该会看到一个“开关”的标志，证明 MCS 端的控制已经就绪。

本地硬件使用的是 Widora 开源硬件，以板载的 WLED 灯为控制目标。板载了 OpenWrt-Chaos Calmer（Linux）操作系统，故本地需要开发一个 javascript 的应用程序连接 MCS 云获取数据，进而本地处理数据。由于本地需要和 MCS 云连接，故本地要用到 mcsjs 模块，在 Widora 上可以通过#npm install mcsjs 提前安装一下。本次应用程序设计的流程图如下：



该程序中，需要先调用 mcsjs 模块的 register 方法将预先获取的 DeviceID 和 DeviceKEY 注册并 TCP 长连接到 MCS 云，进而注册一个监听事件对接收到的数据进行判断。核心就是数据的一个处理。首先本地的打开和关闭 LED 使用 shell 命令即可，具体命令是：

打开 LED：

```
#echo 1 > /sys/class/leds/mediatek:orange:wifi/brightness
```

关闭 LED：

```
#echo 0 > /sys/class/leds/mediatek:orange:wifi/brightness
```

下面贴出所有程序 app.js：

```
var mcs = require('mcsjs');              //使用 mcsjs 模块，用于 TCP 长连接到 MCS 云通信
var exec = require('child_process').exec;      //使用 child_process 模块用于执行 shell 命令
var myApp = mcs.register({
 deviceId: 'DfjtWtoa',              //注册 deviceId
 deviceKey: '7iXqgxkDmBrjQ7eJ',          //注册 deviceKey
 host: 'api.mediatek.cn'            //mcs 域名
```

```
});
console.log('mcs init ok');
myApp.on('switch1', function(data, time) {          //注册 switch1 监听事件用于监听数据
    if(Number(data) === 1){        //如果数据为 1
        console.log('on');              //控制台输出 on
        exec('echo 1 > /sys/class/leds/mediatek:orange:wifi/brightness',
function(error,stdout,stderr){});       //打开 LED
    } else {        //如果数据不为 1
        console.log('off');             //控制台输出 off
        exec('echo 0 > /sys/class/leds/mediatek:orange:wifi/brightness',
function(error,stdout,stderr){});       //关闭 LED
    }
});
```

本地程序写好后，可以使用#node app.js 运行看看有没有错，我的运行结果如下：

root@Widora:~# node app.js

mcs init ok

此时点击页面的开关，本地会交替打印出如下信息，同时也会看到 WLED 亮灭交替变化：

```
on
off
on
off
```

WLED 灭：

WLED 亮：

5、总结

通过使用了 MCS 云平台和 Widora 展示的云端控制 LED 实例，在不足 20 行代码的情况下完成了 C 语言数百行才能做到的事情，并且整个开发没有碰触到 Linux 驱动层或 Linux 本身系统层的难点，在该实例的背后，有相当多的基础服务、基础平台的支撑，才得以使上层开发如此简单。今后的新型物联网嵌入式，开发方式一定是朝着抽象化、简单化、层次化的方向衍变，并且可能会派生出更多类似 node.js 的平台，我也相信国内会有更多的开源爱好者加入到新型物联网嵌入式的开发潮流，为整体软硬件生态做出更多贡献。