使用 **Mongoose** 连接 **ESP32** 与 **AWS IoT** （一）：**ESP8266** 上设置 **AWS IoT**

Phodal Huang

October 24, 2017

目录

原文链接：https://www.wandianshenme.com/play/esp32-mongoose-os-build-iot-application-with-a

一旦你停止对 ESP32 功能列表的赞美，接下来的时候，就是让你的 ESP32-DevKitC 能够做一些比芯片本身更重要的东西。如 AWS IoT 这样的云产品，它是云服务的起点的解决方案，用于与您的 ESP 进行通信和分析数据。

我们的目标是

- 建立与 ESP32 开发板一起工作的开发环境
- 将 ESP32 配置为 AWS IoT Thing，并使其准备好进行通信

步骤 0：概念

什么是 ESP32

一个低功耗、低成本的微控制器单元（MCU），板载 2.4Ghz WiFi 和蓝牙 BLE。目前，单个 ESP32 模块（无扩展）约为 6.50 美元。

ESP32-DevKitC（Amazon，$ 14.99）是 Espressif 推出的一个开发板，旨在促进从设计到市场的 IoT 设备的开发和原型开发。它们采用可在商业应用中使用的核心 SoM（系统模块），并用串行通信芯片、USB 连接器、有线控制编程状态的按钮，电源 LED 和面包板兼容引脚头运动减肥。想进一步了解，查看他的 datasheet。

ESP32 与 Espressif 前身、及带有 WiFi 功能的 MCU 竞争对手之间的差异之一是，其多种安全功能：

- SPI Flash 加密：教程
- 安全启动：教程
- 用于 SHA，AES，RSA 等的硬件加速加密
- 安全无线代码更新：教程
- 支持 ATECC508A 集成：教程

注意：我购买的 ESP32-DevKitC 似乎缺少 SoM 的板载 LED（或者分线板上的电源 LED 指示器是 GPIO 可寻址的 LED 并且不起作用）。

什么是 AWS IoT

用于在互联网连接的设备和云之间提供双向通信的平台。它提供：

- 设备网关（Device Gateway）：使其他设备和应用程序能够与设备进行通信。

- 消息代理（Message Brokering）：用于通信的设备的 pub/sub 机制
- AWS 集成：允许（几乎）与其他 AWS 产品（如 DynamoDB 或 S3 ）的无缝集成
- Shadows：设备状态和被管理数据的持久对象表示。IMHO，让人联想起虚拟 DOM 代表单一设备的想法。

什么是 **Mongoose OS**

一个 “物联网的开源操作系统” 虽然听起来很酷，但是它可以分解成：

- 用于读取和写入 ESP32（或其他支持的设备）的整合工具链
- 用于处理诸如 AWS IoT 和 Google Cloud IoT 之类的云集成的库和工具。
- 用于与设备交互的简单 Web 客户端 IDE。
- 您的设备的 Javascript 支持，包括用于执行常见任务的库，以及将 C 代码（想一下 Arduino 库或 C-only 的助手方法）移植到 Javascript 中的实用程序！
- 准备为 ESP32 和支持的设备闪存自定义固件

条件准备

- ESP32-DevKitC 开发板
- USB A 转 Micro USB 连接器
- AWS CLI
- AWS 账号，其凭证存储在您的本地环境中

步骤 **1**：搭建

搭建你的笔记本

看看您的 ESP32-DevKitC 上的串行芯片（如果您是像我这样的老人，您可能需要打破放大镜），并识别芯片上的第二行文本 - 如果您的是 CP2102，那么您应该下载 Silicon Labs 的驱动程序，否则如果您看到 CH34x，您应该下载相应的 CH34x 驱动程序。

macOS

- Silicon Labs 驱动程序（串行芯片上的 CP2102）：下载地址
- CH34x 驱动程序（串行芯片上是 CH340 或者 CH341）：下载地址

Windows

- Silicon Labs 驱动程序（串行芯片上的 CP2102）：下载地址
- CH34x 驱动程序（串行芯片上是 CH340 或者 CH341）：下载地址

然后，安装相应的驱动程序。

测试安装的驱动程序

确保您的 Micro USB 电缆连接器，已插入 ESP32-DevKitC 上的配对连接器，USB A 连接器已从笔记本电脑拔下。再将 USB A 连接器插回笔记本电脑，并打开一个新的终端：

```
$ ls /dev | grep tty.SLAB
tty.SLAB_USBtoUART
```

注意该命令的输出 - 它将是 Mongoose OS 尝试连接的串行设备的名称。在当前情况下，输出为 tty.SLAB_USBtoUART -> 这将转换为完整的设备路径：

```
/dev/tty.SLAB_USBtoUART
```

如果您没有看到像如上的任何输出，请断开 USB 电缆并重新连接到笔记本电脑，并尝试更通用的命令，如 `$ ls/dev | grep tty`，这次你会看到许多设备; 使用 USB 或 UART 查找设备名称。如果您仍然看不到您的设备，应该还有其他错误。

步骤 2：安装 Mongoose OS

这里假定您已经安装了串行驱动程序。Mongoose OS 是与 USB 串行连接设备（即 ESP32 ）进行交互的命令行工具链和 Web IDE 的组合。

macOS

```
$ curl -fsSL https://mongoose-os.com/downloads/mos/install.sh | /bin/bash
```

Windows

先下载可执行的安装文件：mos.exe

```
$ mos console
```

步骤 3：烧录 ESP32 固件

注意：通过 CLI 烧录是我可以让 Flash 操作能正常工作的唯一方法（macOS Sierra，Chrome 59）。

macOS

```
$ ~/.mos/bin/mos flash esp32 --port /dev/cu.SLAB_USBtoUART
```

注意：cu.DEVICENAME 是表示我们连接到外部设备的方式的呼叫（call-up）设备

Windows

```
$ mos flash mos-esp32
```

```
Loaded default/esp32 version 1.0 (20170801-121654/???)
Using port /dev/cu.SLAB_USBtoUART
Opening /dev/cu.SLAB_USBtoUART @ 115200...
Connecting to ESP32 ROM, attempt 1 of 10...
  Connected
Running flasher @ 460800...
  Flasher is running
Flash size: 4194304, params: 0x0220 (dio,32m,40m)
Deduping...
    16304 @ 0x1000 -> 0
     3072 @ 0x8000 -> 0
    16384 @ 0x9000 -> 12288
  1360000 @ 0x10000 -> 0
   262144 @ 0x190000 -> 61440
Writing...
    12288 @ 0x9000
     8192 @ 0xd000
     8192 @ 0x199000
    53248 @ 0x1a7000
Wrote 81920 bytes in 1.00 seconds (641.50 KBit/sec)
Verifying...
    16304 @ 0x1000
     3072 @ 0x8000
    16384 @ 0x9000
     8192 @ 0xd000
  1360000 @ 0x10000
   262144 @ 0x190000
Booting firmware...
All done!
```

步骤 4：设置 WiFi

注意：ESP32 WiFi 仅支持 2.4GHz。5GHz Wi-Fi 网络将无法正常工作。

macOS

```
$ ~/.mos/bin/mos wifi YOUR_WIFI_SSID YOUR_WIFI_PASSWORD
```

Windows

```
$ mos wifi YOUR_WIFI_SSID YOUR_WIFI_PASSWORD
```

步骤 5：设置 AWS IoT

Mongoose OS 的 CLI 工具提供了一个方便的设置命令，执行以下操作：

- 检查您存储的 AWS 凭据，以查看要使用的帐户
- 为 Mongoose OS 和 ESP32 创建一个用于与 AWS IoT 进行通信的默认策略（AWS IoT Policies）
- 提供通信所需的证书，公共和私钥，并将其写入 ESP32
- 在 AWS IoT 上为你的设备创建一个 “thing”

此时，请确保您的 AWS 凭据存储在您要使用的本地，并且您的 ESP32 已连接到您的笔记本电脑。将以下命令替换为 REGION，并使用 us-east-1 或适当的 AWS 区域。接着，在你的终端，运行：

macOS

```
1 $ ~/.mos/bin/mos aws-iot-setup --aws-region REGION --aws-iot-policy
    mos-default
```

**Windows*

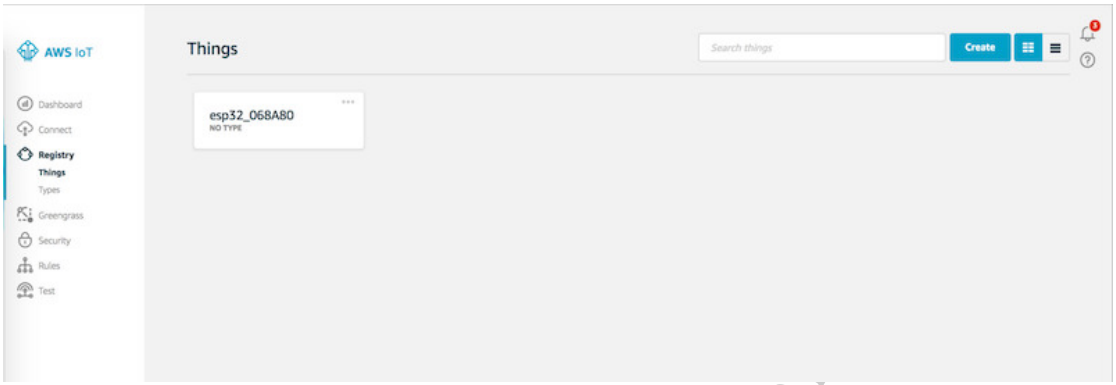
```
1 $ mos aws-iot-setup --aws-region REGION --aws-iot-policy mos-default
```

注意：如果将 mos-default 重命名为另一个策略名称，则可能会遇到错误。尝试使用您想要的名称，在 AWS IAM 控制台上创建一个策略，然后使用其他名称运行此策略。

示例输出如下：

```
Using port /dev/cu.SLAB_USBtoUART
AWS region: us-east-1
Connecting to the device...
Current MQTT confit: {
  "clean_session": true,
  "keep_alive": 60,
  "pass": "",
  "pub": "/response",
  "reconnect_timeout_max": 60,
  "reconnect_timeout_min": 10,
  "server": "axxxxxxxxxxxxz.iot.us-east-1.amazonaws.com:8883",
  "ssl_ca_cert": "ca-verisign-ecc-g2.crt.pem",
  "ssl_cert": "aws-iot-exxxxxxxxf.crt.pem",
  "ssl_cipher_suites": "",
  "ssl_key": "aws-iot-exxxxxxxxf.key.pem",
  "ssl_psk_identity": "",
  "ssl_psk_key": "",
  "sub": "/request",
  "user": "",
  "will_message": "",
  "will_topic": ""
}
Generating certificate request, CN: mos-M3SRGEJDINMOPB12
Generating private key locally
Asking AWS for a certificate...
Certificate ID: bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx8
Certificate ARN: arn:aws:iot:us-east-1:0xxxxxxxxx8:cert/ bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx8
Wrote private key to aws-iot-bxxxxxxxc.key.pem
Wrote certificate to aws-iot-bxxxxxxxc.crt.pem
```

如果您进入 AWS 控制台 -> AWS IoT -> 注册表，您应该会看到像这样注册的设备：



步骤 6：运行 IDE

在终端中运行以下命令：

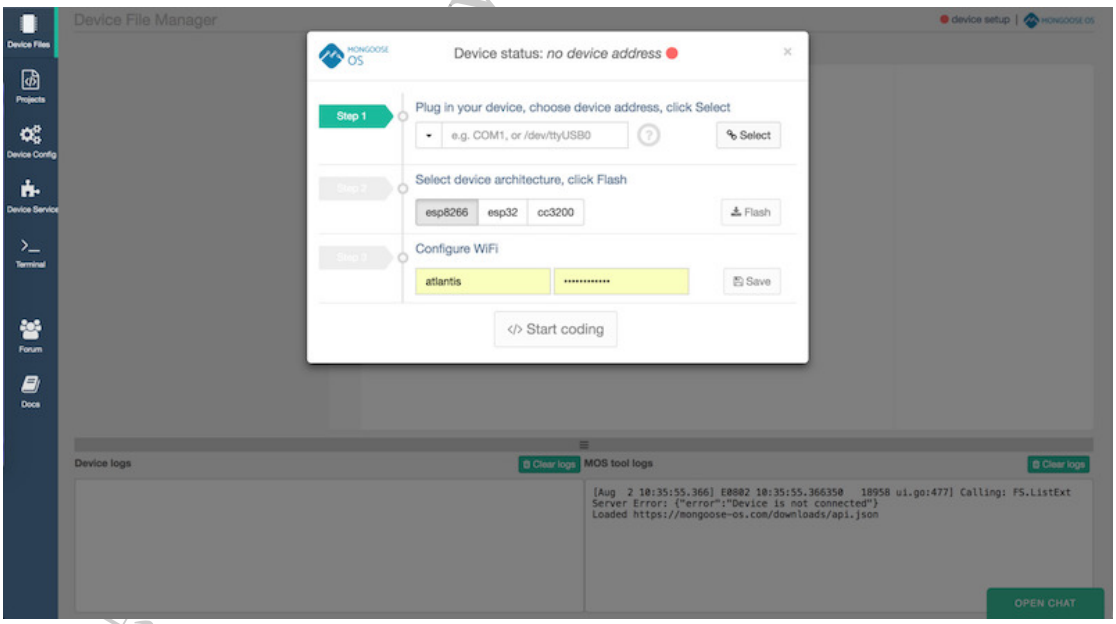
macOS

```
$ ~/.mos/bin/mos ui
```

Windows

```
$ mos ui
```

您的默认浏览器应该，打开一个如下所示的选项卡：



注意：

- Device status（设备状态）报告了 Step 1 中选择的连接设备是否连接到互联网。这并不意味着设备和计算机之间的连接

- Step 1 中的选择按钮不符合我的想法。它确认您在 Step 1 中输入到下拉菜单/文本框中的设备。
- Device logs (设备日志，屏幕底部阴影部分）是串行监视器输出。观看您的设备的通信。
- MOS 工具日志（屏幕的右下角）是工具链（IDE，编译器，闪烁器等）的消息区域

步骤 7：为 **ESP32** 设置 **IDE**

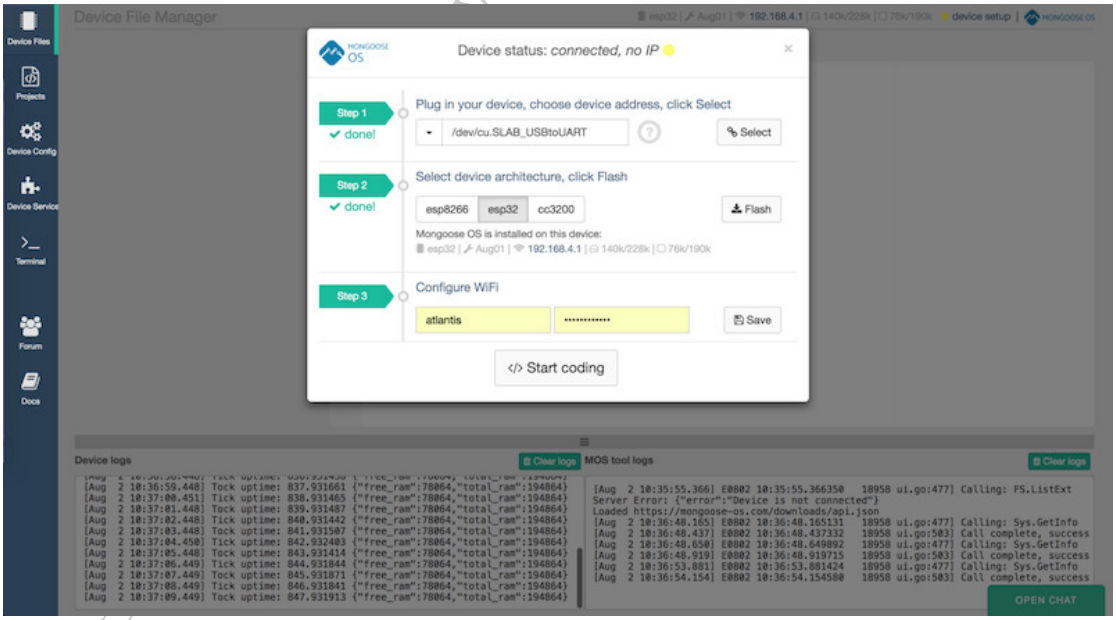
在 Step 1 区域，使用我们从“测试驱动程序安装”（Testing your Driver Installation）中注意到的设备（请记住我们使用了命令：`ls /dev | grep tty.SLAB`），我们将其完整路径输入到下拉列表/文本框中，如下所示：

```
1 /dev/cu.SLAB_USBtoUART
```

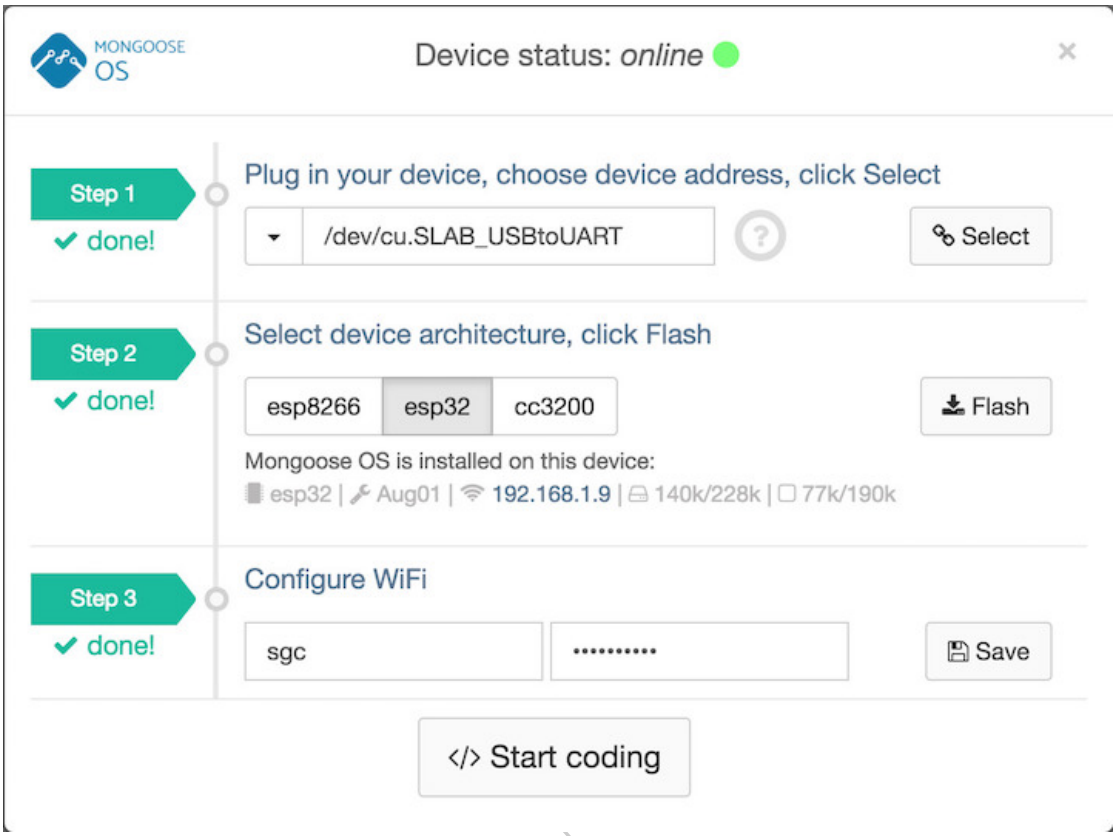
按选择按钮。您应该看到设备状态显示连接状态（有一个绿色或黄色的点）。您还应该在 Step 2 部分中看到一些设备信息。

在 Step 2 区域，选择 esp32。不要点击 **Flash** 按钮。我们已经在之前的一步中烧录过了我们的设备。

如果 Step 1 中的 WiFi 在设备状态中留下了黄色的点和“无 IP”，则可能需要在 Step 3 里修改 WiFi 信息，然后点击保存按钮。



成功！



下一步

现在我们的设备和环境正在运行，我们将会解决 ESP 和云端之间的通信示例。

- 修改 thing 的 Shadow 状态
- 将 Shadow 状态推回给 ESP32
- 订阅 AWS IoT 中的 MQTT 事件

原文链接：https://medium.com/@gomaketeam/connecting-the-esp32-devkitc-and-aws-iot-using-mongoose-os-part-i-11fecd2d86d5

原文链接：https://www.wandianshenme.com/play/esp32-mongoose-os-build-iot-application-with-a