デバイスの種類、調達と実装、関連する規制や法制度

1.1 デバイス

IoTの末端となるデバイスはさまざまなものがあります。センサーなどの半導体、エンドノードやリーフノードと呼ばれる末端の機器も含まれます。サーバのハードウェアもデバイスです。本節ではそれらのデバイスの種類や特徴、関連する規制や法律などの情報について説明します。

坪井 義浩　TSUBOI Yoshihiro

はじめに

筆者は、Seeed Technology Limited（URL https://www.seeedstudio.com）という、中国の広東省深圳（シンセン）に本社を構える会社で、プロダクト担当VP兼日本担当のカントリーマネージャーをしています。本節では、「世界の工場」と呼ばれる中国と、日本でのハードウェアの小ロット設計・生産の経験を生かして、読者の皆さまの役に立つような考え方を紹介します。

プラットフォームの選び方

「IoTのエンドノードを作る」には、まずどのようなプラットフォームを使って試作をするかを決めなければなりません。ここ10年くらいの間にハードウェア分野に訪れたオープンソースムーブメントによって、最近では非常に多くのプロトタイピング向けのプラットフォームが登場しています。

こういったクイックでダーティ（不完全）なプロトタイプは、もちろん実際にデプロイ（設置）して使用するデバイスとは少し異なるものになりがちです。一方、実際にデプロイするデバイスに近づければ近づけるほど、プロトタイプで得られた知見を生かすために手戻りするときの工数が大きくなっていきます。だからといって、実際のデバイスからかけ離れたものでは、プロトタイプによって確認できることが少なくなってしまいます。このあたりのバランスをどうやって取っていくかが、プラットフォーム選びのポイントになります。ここからは、実際にデプロイするデバイスのことを念頭に置きながら、どういったことに留意し、どのようにプラットフォームを選んでいくのがよいのか考えていきます。

どう設置するかを考えよう

まず、作ったデバイスをどこにどうやって設置するのかを考えましょう。

たとえば、ユーザーが身につける、つまりウェアラブルを作るのであれば重量は軽く、薄く小さく、できれば生活防水程度の防水性能が必要だろうということになります。身につけると言っても、ポケットに入れたり首から提げるのではなく、腕に巻くなど直接肌に触れるのであれば、汗のことを考えるとより高い防水性能が求められるでしょう。また、発熱をするようなデバイスは、直接肌に触れるようなウェアラブルでは歓迎されないでしょう。

一方、身につけるデバイスであれば、電池で動かすことは前提になります。とはいえ、ウェアラブルであれば頻繁にデバイスの電池を交換したり充電したりするのも簡単です。

ウェアラブルではなく装置を設置するにしても、さまざまなシチュエーションが考えられます。屋内に設置するのか、屋外に設置するのか、あるいは屋外でも自分たちが管理している建物の近くに設置するのか、都市部に設置するのか、人里離れた場所に置くのかなど考えられます。設置する状況によって許容される大きさは変わりますし、電源の状況も変わります。

屋内で自社が管理する物件であれば、防水性能は求められないでしょうし、コンセントからの常時給電やWi-Fiでインターネットに接続をすることも期待できます（**図1**）。しかし屋外設置だと、防水性能どころか耐候性（屋外は紫外線が当たったり、温度も湿度も変わりますので、防水性能が劣化しがちです）が求められます（**図2**）。

■図1　屋内設置の例

■図2　屋外設置の例

このほかに、昆虫がデバイスに巣を作ったりといった問題や、コンセントからの常時給電が難しくなるといった問題も発生します。また、忘れられがちですが一般的な防水や耐候性だけでなく、特に多雷地域では落雷といった自然現象についても検討をしなければなりません。

屋外でも、人里離れた場所であれば常時給電は非現実的な選択肢になります。電池交換ができる頻度もとても限られてきます。そうなると、長期間運用するためにデバイスの消費電力をできるだけ抑えるだけでなく、どう充電するのかという課題も解決しなければなりません。

もちろん、デバイスをインターネットに接続する手段も、アプリケーションだけでなく設置場所によって変わってきます。デバイスを電池で運用するのであれば、たいていは無線を使うことになるでしょうし、そうなると省電力が求められます。Wi-Fiを使うと手軽にインターネットに接続することができますが、アクセスポイントを用意しなければならなかったり、公衆無線LANであればそれぞれの認証方式に対応する必要が出てきたりします。

Wi-Fiの難点は、SSID（Service Set Identifier）やクレデンシャルなど、デプロイするときの設定が面倒な点です。高速なWi-Fiを使うと消費電力も増えます。一般にワイヤレス技術は帯域が広かったり、カバーするエリアが広かったりすると消費電力が大きくなりがちです。逆に帯域が小さく、通信出来る距離が短いと消費電力が小さいというトレードオフの関係にあります。これらワイヤレス技術の選択については、第2章を参考にしてください。

先のことを考えた選択を

趣味で作る一点物であれば気にすることはないのですが、ビジネスで展開するIoTデバイスでは、継続的に調達できるかどうかも重要です。プロトタイプ用のプラットフォームは、あくまでプロトタイプ用であり、実際に展開するうえでは要求される速さに応じてデバイスを製造したり用意できなければなりません。また、プロトタイプ用のプラットフォームの多くは、継

1.2 ファームウェア設計

ここでは既存の設計手法よりも発展したIoT時代のファームウェア設計と実装について紹介します。その利点はどこにあるのか、そして、どのようなハードルがあるのか見ていきます。後半では、実際の設計について5つの項目を取り上げ、解説していきます

松下 享平　MATSUSHITA Kohei

はじめに

昨今はIoTのマーケットの可能性に触発されて、ArduinoやRaspberry Piだけでなく、IoTでよく使用される無線通信を最初から搭載した製品も多数登場しています。

たとえばWi-FiとBLE（Bluetooth Low Energy）をワンチップに搭載したESP-WROOM-32、スマートフォンで使われているLTE（Long Term Evolution：高速無線通信技術）のカテゴリ1という規格に対応したセルラーモデムとマイコンがワンパッケージになったWio LTEなどがあります（**図1**）。

前節でも紹介したように、プロトタイプだけでなく量産を行う場合においてもさまざまな選択肢が存在しています。

ファームウェアとは

デバイスを大きく分類してみると、「センサー素子やアクチュエータ」と「マイコン」の2つから構成されています（『IoTエンジニア養成読本』の第3章参照[注1]）。これらを調達して組み立てればIoTにおける「モノ」として利用できるかというと、やはり不足しているものがあります。マイコンがセンサー素子やアクチュエータといったデバイスを制御するためのプログラムが必要なのです。

このような、マイコン上でデバイスを動かすプログラムのことを**ファームウェア**と呼びます（**図2**）。これまでファームウェアはセンサー素子などを制御できればよかったのですが、IoT時代においてはその役割や実装の考え方についても変わってきています。本節ではIoT時代におけるファームウェアの設計や実装について解説していきます。

注1　片山暁雄、松下享平、大槻健、大瀧隆太、鈴木貴典、竹之下航洋、松井基勝（2016）『IoTエンジニア養成読本』技術評論社

■図1　マイコンとして利用可能な製品の一覧


Arduino
（アルディーノ）


Raspberry Pi Zero
（ラズベリーパイゼロ）


ESP-WROOM-32


Wio LTE
（ワイオエルティーイー）

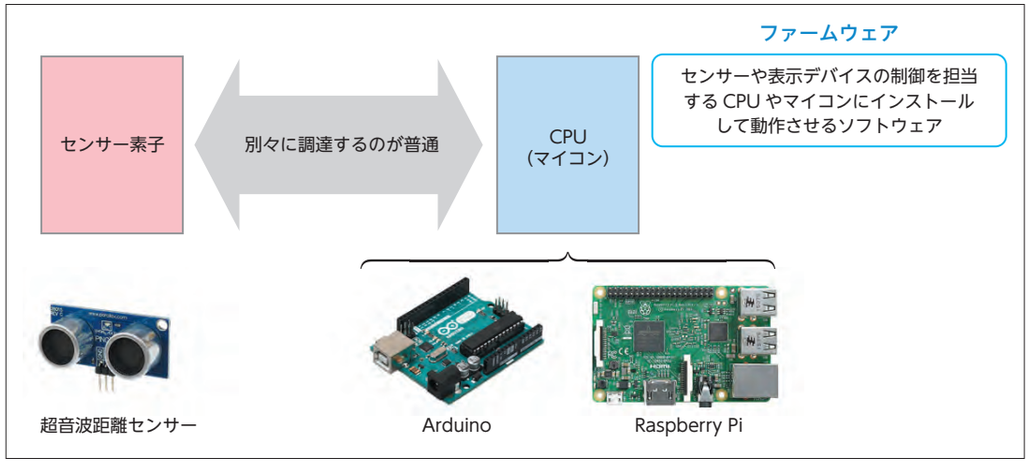
IoT時代のファームウェア開発

ファームウェアについて見ていく前に、改めてIoTがどのようなインパクトを与えたのか再検討してみましょう。

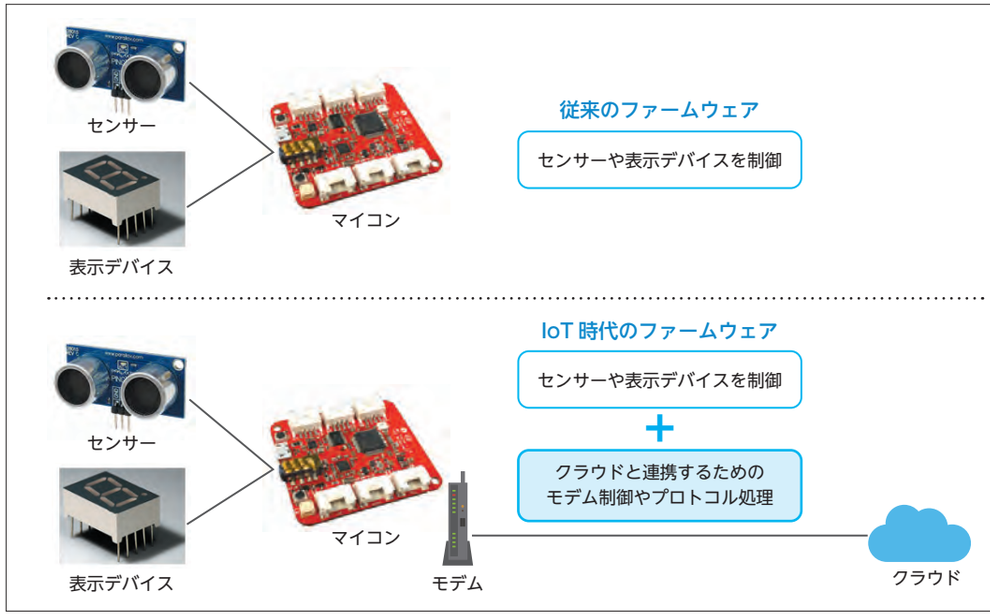
本節ではモノが主役であるので、モノを主語とすると、IoTとは「モノの能力や価値をクラウドの力で向上させる技術」と言えます。

こういった背景を含めてIoT時代におけるファームウェアについて考えると「センサー素子やアクチュエータといったデバイスを制御」という本来の役割に加え「クラウドの力を引き出す」という役割が必要とされるわけです（**図3**）。

■図2　センサー素子やアクチュエータとマイコンの関係



■図3　ファームウェアの役割の移り変わり



ケーススタディ1：WHILL株式会社

5.1 消費電力の最適化

本章では、SORACOMプラットフォーム上で稼働しているIoT事例を4つ取り上げています。最初の事例は消費電力の効率化をテーマに、WHILL株式会社のケースを紹介します。

はじめに

最初に取り上げるのは、消費電力の最適化のお話です。バッテリーで駆動するデバイスはこの課題と常に戦っていく宿命にあります。駆動時間を長くするには、バッテリー容量を大きくすること、消費電力を減らすことの2つのアプローチがあります。

前者は物理的により大きなバッテリーを搭載するか、搭載する電池の数を増やすことになります。やればやるだけデバイスは大きくなり重くなっていきます。容量を増やせば、ほぼリニアに駆動時間も延びるのでシンプルなアプローチではありますが、デバイスの大きさや重さの制約により、無制限にこれが許されることはありません。したがって、後者の消費電力をいかにして減らすのか、つまり電力の効率的な利用が重要なアプローチになってきます。

では、消費電力を減らすにはどうしたらいいでしょうか？ いろいろなアプローチがありますが、ここではWHILL株式会社（以下、WHILL社）が取り組んだ、通信部分における消費電力の最適化の事例を紹介します。

パーソナルモビリティ「WHILL」における省電力化のチャレンジ

パーソナルモビリティ「WHILL」はバッテリーで駆動する、スタイリッシュなデザインに洗練された使い心地と直感的な操作性を兼ね備えた、新しいパーソナルモビリティです（**図1**）。

WHILL社ではカスタマーサポートの効率化のために、稼働中のモビリティの状態を遠隔で、AWS上のサーバに収集する方法を検討していました。しかし、通信をすればするほどバッテリーの消費量は増え、モビリティの稼働時間が削られてしまうという課題があり、これを解決するためにSORACOM Beamを採用しました。SORACOM Beamは、SORACOM Airから利用できるセキュアなリバースプロキシで、デバイスからHTTP、TCP、UDPなどの暗号化されていないデータを受け取り、HTTPSやTCPSにそのペイロードを載せ替えてバックエンドのサーバに届けてくれるサービスです（**図2**）。

SORACOM Beamによる暗号化オフロード

SORACOM Beamは、2つの方向から通信時のデバイスのバッテリー消費を抑えてくれます。

■図1　パーソナルモビリティ「WHILL」

1つ目は、デバイス側でのペイロードの暗号化の必要性がなくなるということです。SORACOM Beamは、beam.soracom.ioというエンドポイントでデバイスからのデータを受け取り、あらかじめユーザーが設定した転送先に、同じく設定されたプロトコルに載せ替えて転送するリバースプロキシです。

SORACOMプラットフォームとWHILL社のサーバの間はインターネットを介した通信が必要で、この区間においてデータを暗号化等により保護する必要がありました。ですが、この暗号化処理をデバイスのCPUに行わせてしまうと消費電力が増えてしまいます。

WHILL社のケースでは、平文のデータをUDPデータグラムに載せて送信し、暗号化（HTTPSへの変換）をSORACOM Beamに任せています。この方式を採用することによりデバイスは暗号化処理から解放され、その分のバッテリー消費を節約しているのです。デバイスからSORACOMプラットフォームの間は携帯電話キャリアが提供するセキュアなセルラーネットワークおよび専用線によって結ばれているため、この区間はデータそのものの暗号化は不要です[注1]。

注1　もちろん、それでもポリシー上の制約であったり、アプリケーションの仕様上、暗号化してデータを通すことにはなんの問題もありません。

SORACOM Beamによるデータ送信量の最小化

さらにこのケースでは、デバイスが使うプロトコルにUDPを選択することにより、プロトコルのオーバーヘッドを最小化しています。無線通信の場合、通信量が増えると電波を発信する量も増え、それがバッテリー消費の増大につながります（**図3**）。

図3は、約10バイトのペイロードを送信するときの総データ送信量を比較したものです。HTTPSよりはHTTP、HTTPよりはTCP、TCPよりはUDPのほうがプロトコルのオーバーヘッドが少なくなります。WHILL社では、最もオーバーヘッドの小さいUDPを選択することによって、電波を発信する量を減らし、バッテリー消費を抑えています。もちろん、UDPは再送制御はありませんし、大きなメッセージを載せることもできません。1つのメッセージサイズは最大で約1500バイト弱に収める、1つのメッセージが到達しなくても次のメッセージで補完ができるようにするなど、プロトコルの制約をアプリケーションでカバーしてやる必要があります。

なお、この図の比較はあくまで新しくセッションを開いて通信をした場合の、いわゆる「最悪時通信量」での比較です。実際にはTCPセッションの再利用による通信量の最適化なども可能なので、ケースバイケースでアプリケーション開発とのトレードオフを勘

■図2　SORACOM Beam