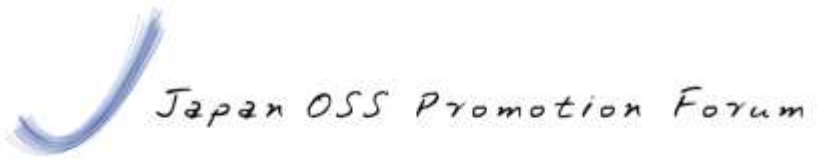
小型ハードウェアモジュールへのmruby適用試行

2018年2月27日

Japan OSS Promotion Forum 2018

株式会社日立ソリューションズ 技術統括本部 IT技術推進センタ

三好 秀徳



目次

1. Ruby / mruby の紹介とアプリ部会とのかかわり
2. 実装したmrubyアプリの紹介
3. 単純なmrubyアプリの開発
4. センサーデータを取得するmrubyアプリの開発
5. まとめ

目次

1. Ruby / mruby の紹介とアプリ部会とのかかわり
2. 実装したmrubyアプリの紹介
3. 単純なmrubyアプリの開発
4. センサーデータを取得するmrubyアプリの開発
5. まとめ

1-1.Rubyとは？

- オープンソースで開発されているプログラミング言語
  - BSDライセンス、Rubyライセンスのデュアルライセンス
- まつもとゆきひろ氏を中心に多くの日本人が開発
  - 日本語ドキュメントが多数存在
  - Ruby関連コミュニティが日本各地に存在
- Webフレームワーク『Ruby on Rails』の実装言語
  - 一躍Rubyが人気言語に
  - 世界で使われているプログラミング言語 20位以内
    http://www.tiobe.com/tiobe-index/

1-2.mrubyとは？（1）

- Rubyと同じ文法を持つ組込み向けプログラミング言語
- オープンソースとして開発
  - https://github.com/mruby/mruby
- 処理系（CPUやOS）に非依存
- 既存C言語資産との互換性あり
  - mrubyからCアプリの呼び出し、C言語からmrubyアプリの呼び出しが可能
- 言語機能の追加だけでなく削除も自由に可能
  - 実行バイナリサイズや使用メモリの低減が可能

1-3.mrubyとは？（2）

- 適用事例多数
  - IIJ社のインターネットルーター『SA-W1』
    - 文字列処理が得意な点を利用し、設定管理などの機能を容易に実装
  - QPS研究所による小型人工衛星システム
    - 処理系（CPU、OS）に非依存な点を利用し、高汎用性を実現
  - Webサーバの機能拡張『mod_mruby』『ngx_mruby』
    - Rubyがもつ高い可読性を利用し、複雑な設定ファイルの記述を読みやすい形で表現
    - C言語への組み込みが可能である点を利用し、高速な動作を実現

1-4.これまでのアプリ部会の取り組み

- 『mrubyとCRubyとの性能比較』
  - Japan OSS Promotion Forum 2015 冨田昌宏氏（株式会社富士通）

    http://ossforum.jp/jossfiles/JOSSPF-2015-7-3.pdf
- 『IoTへのRuby/mruby適用試行』
  - Japan OSS Promotion Forum 2016 廣田哲也氏（株式会社シーイーシー）

    http://ossforum.jp/jossfiles/05_01_application.pdf
- 『IoT適用・検証報告』
  - Japan OSS Promotion Forum 2017江洲信也氏（株式会社エルエスアイ開発研究所）

    http://ossforum.jp/jossfiles/SS6.pdf

1-5.なぜmrubyを使うのか？

- 学習コストが低い
  - 使い慣れたRubyと文法が同じ
- 安全
  - 細かいメモリ確保・解放処理を書く必要なし
- 拡張性
  - C言語で作られたライブラリをmrubyから操作できる
  - mrubyのアプリケーションをC言語側から操作できる

1-6.本年度の実施内容と紹介内容

- 小型ハードウェアモジュール上で動くmrubyアプリを開発
  - ESP-WROOM-32上で動くmrubyアプリの開発
  - ESP-WROOM-32のライブラリをmrubyから利用

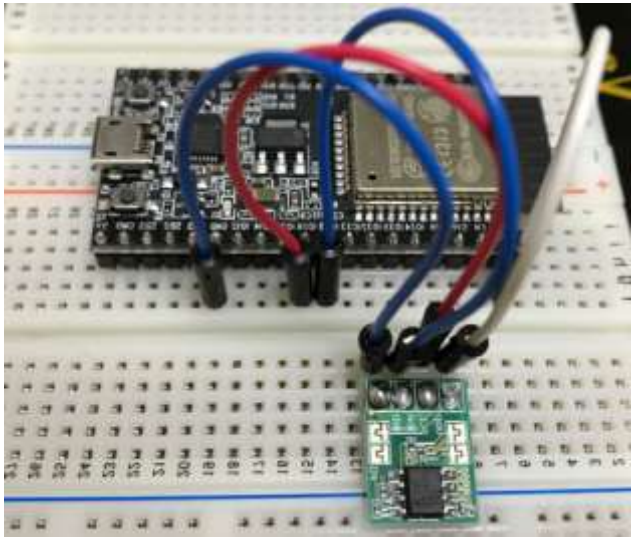
- mrubyアプリ開発における知見を紹介
  - 開発の進め方
  - C言語ライブラリをmrubyから利用する方法

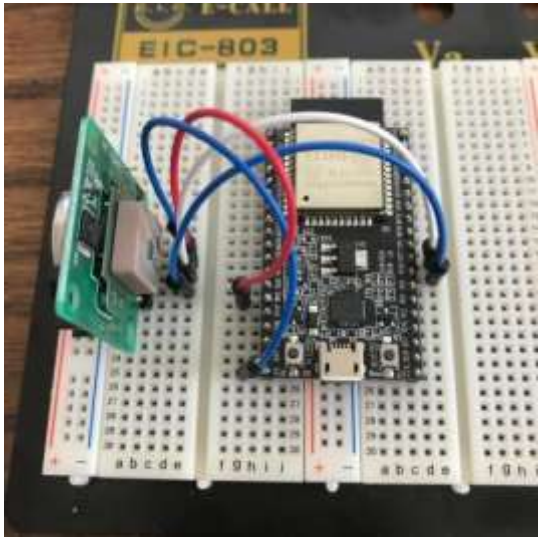
目次

1. Ruby / mruby の紹介とアプリ部会とのかかわり
2. 実装したmrubyアプリの紹介
3. 単純なmrubyアプリの開発
4. センサーデータを取得するmrubyアプリの開発
5. まとめ

2-1.実装内容の概要

- 小型ハードウェアモジュール上に接続したセンサーデータを出力するアプリ
  - mrubyは動くが、Rubyは動かないぐらいのリソースを持つ小型ハードウェアモジュール





2-2.小型ハードウェアの選定

ESP-WROOM-32を選定

項目	値（ESP-WROOM-32）
CPU	Xtensa LX6 160～240 MHz 2コア
メモリ	520KB
Flash	内蔵（4MB）
Network	802.11 b/g/n + Bluetooth 4.2
USB 2.0	なし
消費電力	80mA
OS	FreeRTOS

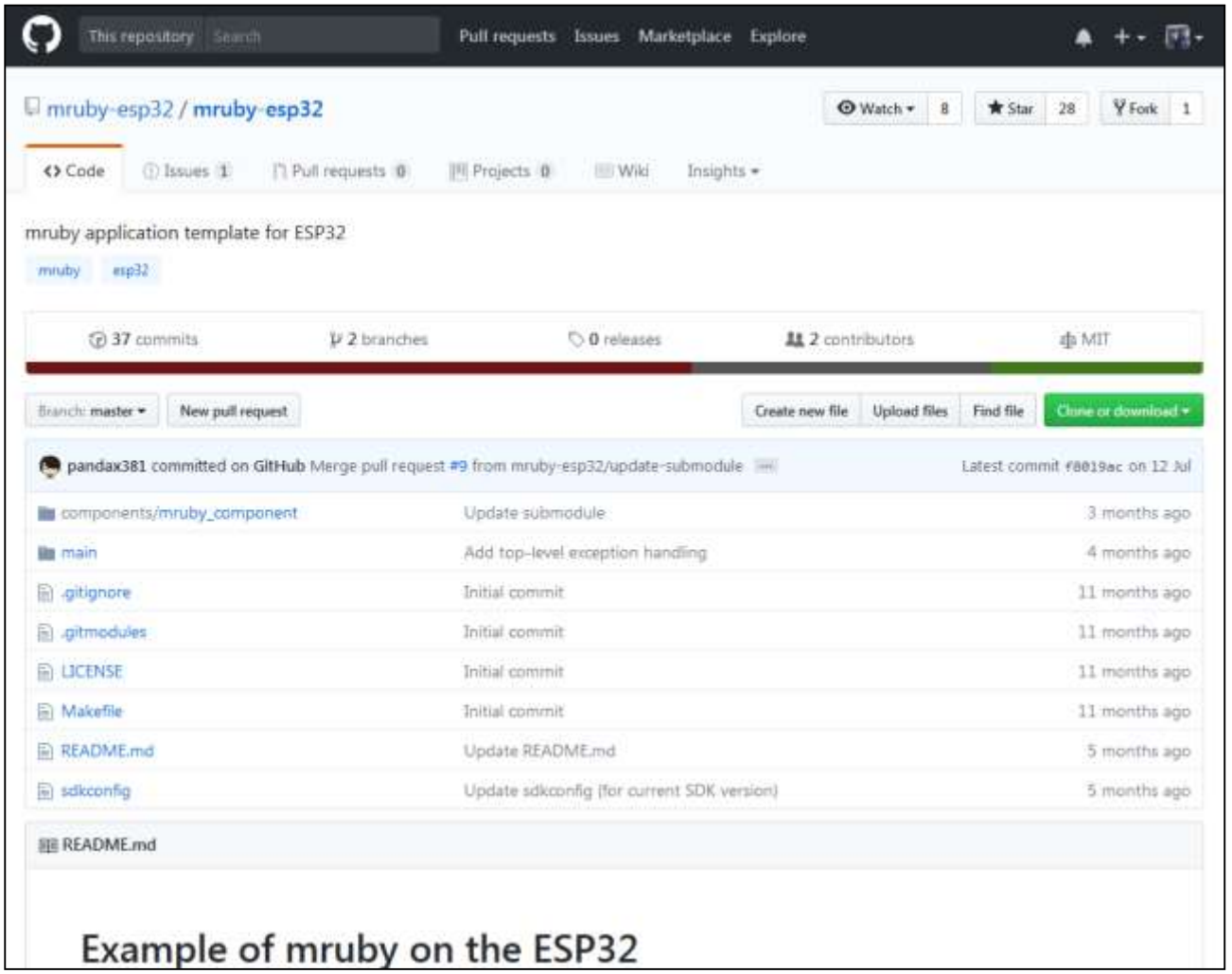
選定理由

- mrubyのフットプリントより大きいメモリを搭載している
- WiFi + Bluetooth内蔵
- 工事設計認証（技適）番号：211-161007

目次

1. Ruby / mruby の紹介とアプリ部会とのかかわり
2. 実装したmrubyアプリの紹介
3. 単純なmrubyアプリの開発
4. センサーデータを取得するmrubyアプリの開発
5. まとめ

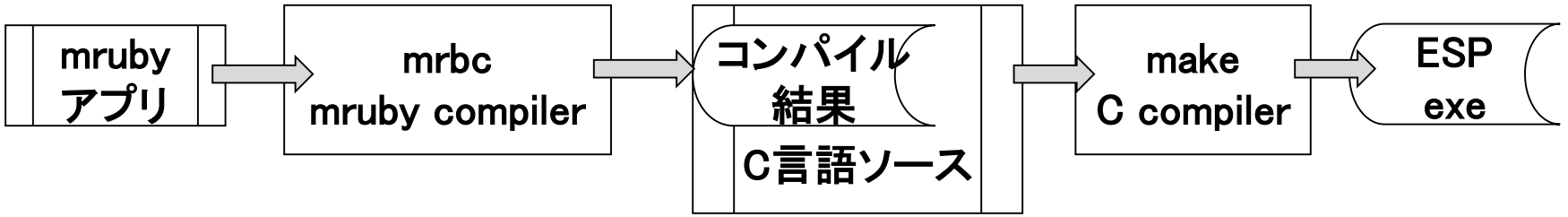
3-1. mruby-esp32



https://github.com/mruby-esp32/mruby-esp32

3-2. mruby-esp32の動作概要

- ESP-WROOM-32上でmrubyを動かすプロジェクト
- 動作原理は単純
  1. mrubyアプリをコンパイルし、C言語の配列として出力する
  2. コンパイル結果をESP-WROOM-32用アプリのC言語ソースで読み込む
  3. C言語ソースをコンパイルしてESP-WROOM-32用バイナリを作成する
- mrubyに手を加えていないので、mruby自身のバージョンアップも容易



目次

1. Ruby / mruby の紹介とアプリ部会とのかかわり
2. 実装したmrubyアプリの紹介
3. 単純なmrubyアプリの開発
4. センサーデータを取得するmrubyアプリの開発
5. まとめ

4-1. 開発の進め方

1. 各種センサーデータを取得してmruby上で扱う

   ① 温度センサーの場合

   ② GPSセンサーの場合

4-1. 開発の進め方

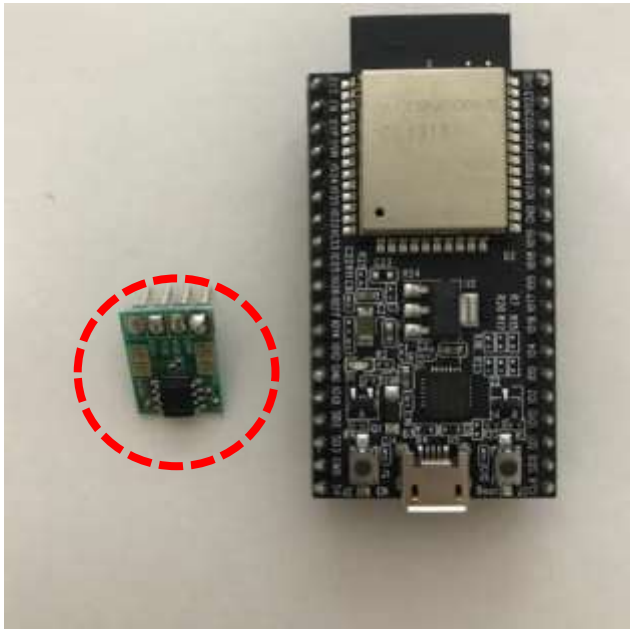
1. 各種センサーデータを取得してmruby上で扱う

   ① 温度センサーの場合

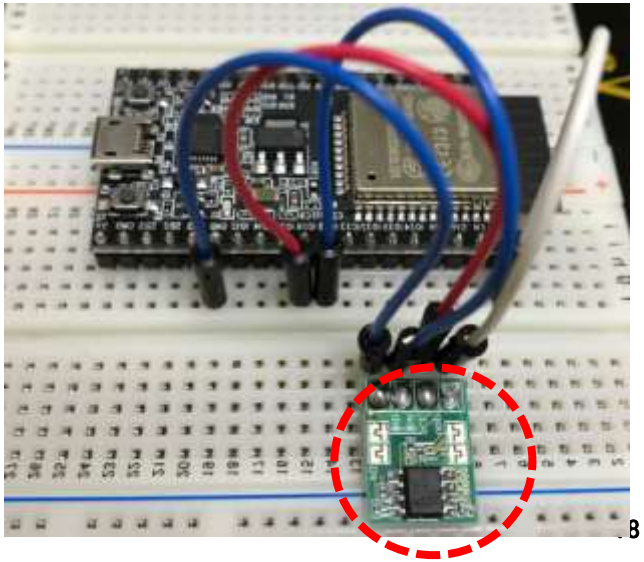
   ② GPSセンサーの場合

4-2. 温度センサーとI²Cの紹介

- ANALOG DEVICES社製温度センサー
  - I²C（アイ・スクエアド・シー）互換インタフェースを持ち、I²Cバスに測定温度データを出力
  - 計測温度データに対してビット演算などを行うことで摂氏データを取得することができる



- I²Cはシリアル通信の方式の一つ
  - センサーとESP-WROOM-32を接続する線のうち真ん中2本の線で通信を実現



4-3. 温度センサーデータを利用するには

- I²C（アイ・スクエアド・シー）通信でデータを取得する必要がある
  - Raspberry Piではi2c-devを設定ファイルに追加するだけ
    - 既存のmrbgemsはLinux上での動作を前提としていて使えない
  - ESP-WROOM-32ではC言語APIを使ってI²C通信でデータを取得できる

- そこでmrbgemsを作成
  - https://github.com/miyohide/mruby-esp32-i2c/tree/add_read
  - I²C通信でデータを取得する部分だけを実装
  - 得られたデータを加工・編集する処理はmrubyアプリ側で実装

4-4. mrbgemsの紹介

mrbgems

```
static mrb_value
mrb_esp32_i2c_init(mrb_state *mrb, mrb_value self)
{
（省略）
}
```

```
static mrb_value
mrb_esp32_i2c_receive(mrb_state *mrb, mrb_value self)
{
（省略）
}
```

I²C通信関連の処理を
C言語で実装

C言語の関数と
mrubyのメソッドとを
紐付け

```
void
mrb_mruby_esp32_i2c_gem_init(mrb_state* mrb)
{
  mrb_define_method(mrb, i2c, "_init", mrb_esp32_i2c_init, MRB_ARGS_REQ(7));
  mrb_define_method(mrb, i2c, "receive", mrb_esp32_i2c_receive, MRB_ARGS_REQ(2));
（以下略）
}
```

4-5. mrubyアプリの実装（1）

mrubyアプリ

```
class ADT7410
  def receive
    word_data = @i2c.receive(2, @addr)
    first_bit = word_data[0].unpack("C*").first
    second_bit = word_data[1].unpack("C*").first
    ((first_bit << 8 | second_bit) >> 3) * 0.0625
  end
end

i2c = I2C.new(I2C::PORT0, scl: I2C::SCL1, sda: I2C::SDA1).init(I2C::MASTER)

adt = ADT7410.new(i2c)
# 初期化処理等は省略
100.times do
  puts "#{Time.now} temp=#{adt.receive}"
  ESP32::System.delay(1000)
end
```

I²C通信の汎用的な部分をmrbgems化

4-6. mrubyアプリの実装(2)

mrubyアプリ

```
class ADT7410
  def receive
    word_data = @i2c.receive(2, @addr)
    first_bit = word_data[0].unpack("C*").first
    second_bit = word_data[1].unpack("C*").first
    ((first_bit << 8 | second_bit) >> 3) * 0.0625
  end
end

i2c = I2C.new(I2C::PORT0, scl: I2C::SCL1, sda: I2C::SDA1).init(I2C::MASTER)

adt = ADT7410.new(i2c)
# 初期化処理等は省略
100.times do
  puts "#{Time.now} temp=#{adt.receive}"
  ESP32::System.delay(1000)
end
```

センサー固有の処理は
mrubyで実装

4-7. mrubyアプリの実行例

```
（省略）
I (1551) cpu_start: Pro cpu start user code
I (1610) cpu_start: Starting scheduler on PRO CPU.
I (2202) mruby_task: Loading binary...
Thu Jan 01 00:00:00 1970 temp=27.4375
Thu Jan 01 00:00:01 1970 temp=27.4375
Thu Jan 01 00:00:02 1970 temp=27.4375
Thu Jan 01 00:00:03 1970 temp=27.4375
Thu Jan 01 00:00:04 1970 temp=27.375
Thu Jan 01 00:00:05 1970 temp=27.4375
Thu Jan 01 00:00:06 1970 temp=27.4375
Thu Jan 01 00:00:07 1970 temp=27.4375
Thu Jan 01 00:00:08 1970 temp=27.375
Thu Jan 01 00:00:09 1970 temp=27.4375
Thu Jan 01 00:00:10 1970 temp=27.4375
```

温度が正しく取れている

4-8. 開発の進め方

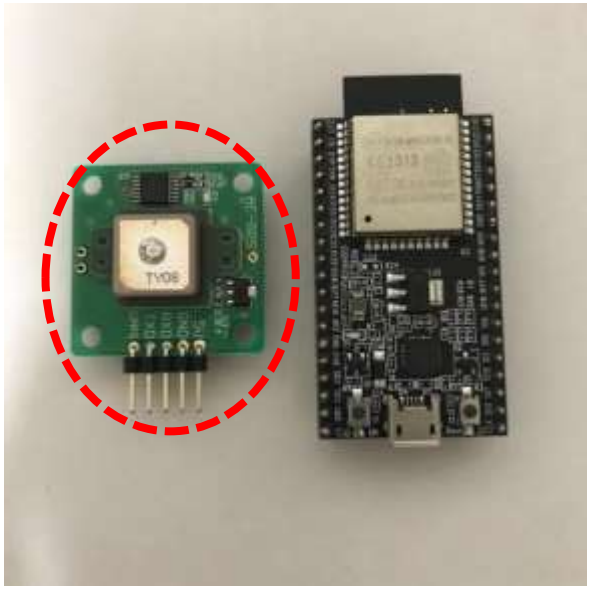
1. 各種センサーデータを取得してmruby上で扱う

   ① 温度センサーの場合

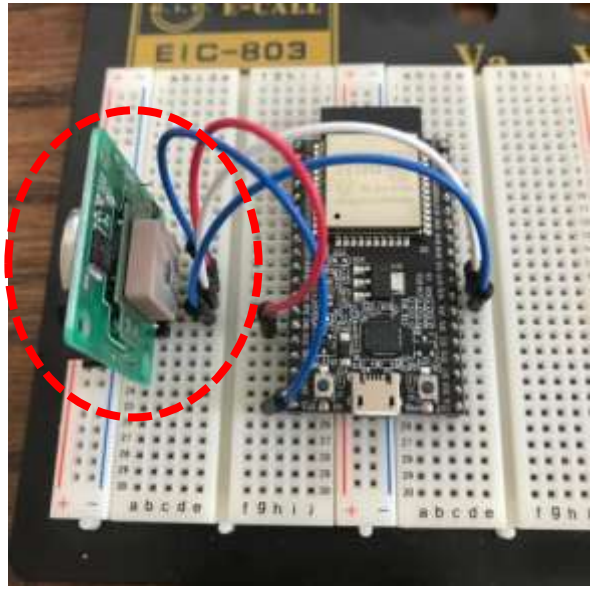
   ② GPSセンサーの場合

4-9. GPSセンサーとUARTの紹介

- 太陽誘電社製 GPSセンサー
  - UARTシリアル通信にてGPSデータ（NMEA0183形式）を取得可能
  - NMEA0183形式で得られたデータを加工・編集することで測定位置の緯度・経度データを取得することができる



- UARTはシリアル⇔パラレル変換を行う集積回路の名称
  - 汎用のシリアル通信機能をもち、定められた規格のシリアル通信でデータ通信が可能



4-10. GPSセンサーを利用するには

- UARTシリアル通信でデータを読み込む必要がある
  - Raspberry Piでは配線するだけで/dev/ttyAMA0にデータが出力される
    - 既存のmrbgemsはLinux上での動作を前提としているため使えない
  - ESP-WROOM-32ではC言語APIが提供されている

- そこでmrbgemsを作成
  - https://github.com/miyohide/mruby-esp32-gps
  - UARTシリアル通信でデータを取得する部分のみを実装
  - 得られたデータを加工するのはmrubyアプリにて実装する

4-11. mrbgemsの紹介

mrbgems

```
static mrb_value
mrb_esp32_gps_init(mrb_state *mrb, mrb_value self) {
（省略）
}
```

```
static mrb_value
mrb_esp32_gps_dogps(mrb_state *mrb, mrb_value self) {
（省略）

}
```

UARTシリアル通信関連の処理をC言語で実装

```
void mrb_mruby_esp32_gps_gem_init(mrb_state* mrb) {
（省略）
  mrb_define_module_function(mrb, gps, "init", mrb_esp32_gps_init,
    MRB_ARGS_NONE());
  mrb_define_module_function(mrb, gps, "doGPS", mrb_esp32_gps_dogps,
    MRB_ARGS_NONE());
}
```

C言語の関数とmrubyのメソッドとを紐付け

4-12. mrubyアプリの紹介

mrubyアプリ

```
include ESP32::GPS

init

1000.times {
    puts doGPS
}
```

mrbgems化した
UARTシリアル通信の部分を
呼び出し

4-13. mrubyアプリの実行

```
I (1950) uart: queue free spaces: 10
$GPGGA,044241.000,xxxx.xxxx,N,yyyy.yyyy,E,1,5,2.56,105.7,M,39.5,M,,*5B
$GPGSA,A,3,193,06,19,17,09,,,,,,,,2.74,2.56,0.97*36
$GPRMC,044241.000,A,xxxx.xxxx,N,yyyy.yyyy,E,0.37,98.78,081017,,,A*54
$GPZDA,044241.000,08,10,2017,,*5C
$GPGGA,044242.000,xxxx.xxxx,N,yyyy.yyyy,E,1,5,2.56,105.7,M,39.5,M,,*5A
$GPGSA,A,3,193,06,19,17,09,,,,,,,,2.74,2.56,0.97*36
$GPRMC,044242.000,A,xxxx.xxxx,N,yyyy.yyyy,E,0.30,98.78,081017,,,A*52
$GPZDA,044242.000,08,10,2017,,*5F
$GPGGA,044243.000,xxxx.xxxx,N,yyyy.yyyy,E,1,5,2.56,105.7,M,39.5,M,,*58
$GPGSA,A,3,193,06,19,17,09,,,,,,,,2.74,2.56,0.97*36
$GPRMC,044243.000,A,xxxx.xxxx,N,yyyy.yyyy,E,0.27,98.78,081017,,,A*56
$GPZDA,044243.000,08,10,2017,,*5E
$GPGGA,044244.000,xxxx.xxxx,N,yyyy.yyyy,E,1,5,2.56,105.7,M,39.5,M,,*5E
$GPGSA,A,3,193,06,19,17,09,,,,,,,,2.74,2.56,0.97*36
$GPGSV,4,1,13,193,83,295,23,06,71,081,27,
$GPGSV,4,2,13,19,36,181,25,05,28,270,,07,19,107,20,
$GPGSV,4,3,13,23,15,042,,30,12,139,,13,06,205,,12,05
$GPGSV,4,4,13,29,03,327,*45
```

緯度・経度が取れている
（xxxx.xxxx / yyyy.yyyyの部分）

目次

1. Ruby / mruby の紹介とアプリ部会とのかかわり
2. 実装したmrubyアプリケーションの紹介
3. 単純なmrubyアプリの開発
4. センサーデータを取得するmrubyアプリの開発
5. まとめ

5-1. まとめ

- 小型ハードウェアモジュール上でmrubyアプリケーションを動かす方法を紹介
- 小型ハードウェアモジュール用C言語ライブラリをmrubyから扱うための開発方法を紹介
- mrbgemsを作成するときに役立つ資料
  - mrubyのGitHubリポジトリにあるdocディレクトリ
    - https://github.com/mruby/mruby/blob/master/doc/guides/mrbgems.md
  - CRubyのGitHubリポジトリにある「Rubyの拡張ライブラリの作り方」
    - https://github.com/ruby/ruby/blob/trunk/doc/extension.ja.rdoc
  - mruby-mrbgem-template
    - https://github.com/matsumotory/mruby-mrbgem-template

5-2. 現状の取り組みと課題

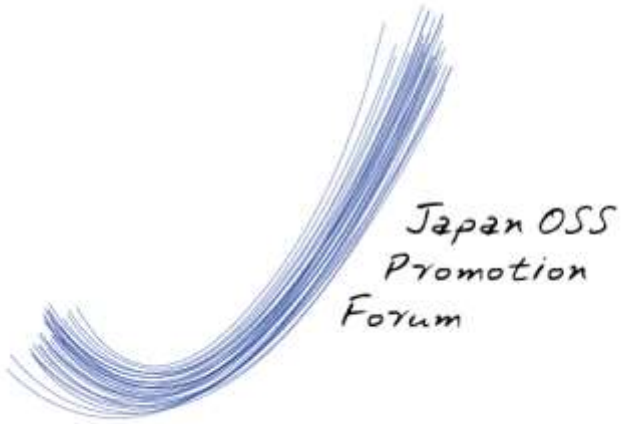
- ネットワーク機能を使ってデータを外部送信
  - https関係のライブラリー不足により既存mrbgemsのコンパイルに失敗
  - ESP-WROOM-32の機能を使って何とか解決できないか調査中

```
（省略）
CC    build/mrbgems/mruby-polarssl/src/polarssl.c -> build/esp32/mrbgems/mruby-polarssl/src/polarssl.o
/Users/miyohide/work/mruby-esp32/components/mruby_component/mruby/build/mrbgems/mruby-
polarssl/src/polarssl.c:20:23: fatal error: sys/ioctl.h: No such file or directory
compilation terminated.
/Users/miyohide/work/mruby-compilation terminated.
rake aborted!
Command Failed: （省略）
make[2]: *** [all] Error 1
make[1]: *** [build] Error 2
make: *** [mruby_component-build] Error 2
```

5-3. Plato

- ハードウェアごとに製品仕様を調べて、mrbgemsを作成するのは大変
  - mruby IoTフレームワーク “Plato”（プラトン）

http://plato.click/



Ruby on Railsは、David Heinemeier Hansson氏の米国およびその他の国における商標または登録商標です。
LinuxはLinus Torvalds氏の米国、日本およびその他の国における登録商標または商標です。
Raspberry Piは、英国Raspberry Pi財団の米国およびその他の国における商標または登録商標です。
Cortexは、ARM Limitedの米国およびその他の国における商標または登録商標です。
その他、記載されている会社名、商品名は、各社の登録商標または商標です。