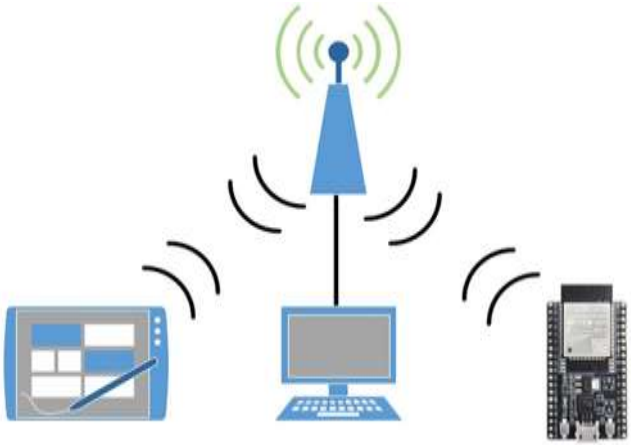
音声認識を利用した
ハードウェア制御システムの構築



島根職業能力開発短期大学校

電子情報技術科

1　はじめに

この教材利用対象は、専門課程2年生後半を想定しており、ハードウェア、ソフトウェア(プログラム)、ネットワークについて習得を行った後としています。

専門課程では 1 年生から 2 年生前半までは、ハードウェア、ソフトウェア、ネットワークなどを単独に理解していくカリキュラムとなっています。ただ、組込み関連業種において、これらを組み合わせて利用することが求められています。そこで、教材開発を行うにあたりこれらの技術を組み合わせ1つのシステム開発に意欲的に取り組める教材を開発したいという思いがあり、この教材を開発することとしました。

併せて、出来るだけ校内にあるものを活用するよう心がけ、最小の費用で実習を行えるよう検討しました。

1.1　教材開発の目的

ハードウェア、ソフトウェア、ネットワークを組み合わせた製品は、IoT(Internet of Things)として、企業内だけでなく、家庭にも浸透し始めています。学生がこのようなシステム開発を体験することで、一連の技術の連携がどのようになっているか実習を通じて理解を深めることを目的としました。

その中で、注目したのが音声認識システムになります。最近では、スマートスピーカー(Apple HomePod、Google Home、Amazon Alexa、Line Clova など) をはじめとした音声認識を利用した製品が身近な環境において活用されています。今後の状況についても総務省から公開されています。

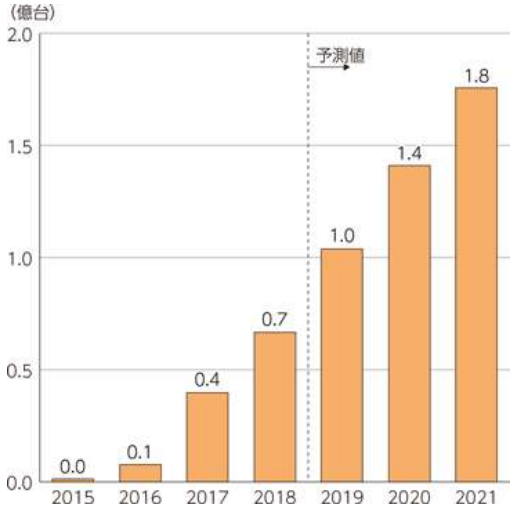


図 1:世界の AI スピーカー(スマートスピーカー)出荷台数の推移及び予測　※1
令和元年版　情報通信白書　レイヤー別にみる市場動向　より

図1のように、音声認識については、今後の発展が期待できる分野となります。

実現するためには、ソフトウェア、ネットワークの連携が必要であり、そこにハードウェアを組み合わせることにより、関連する理解をより深めることができます。さらに、ハードウェア制御だけでなく、その先には、音声認識自体の開発に利用されている人工知能関連について話しを進めていくことが可能になるはずです。また、筆者の施設内の学生にも音声認識システムに関して興味を示す者も多くいました。

音声認識そのものを制作することは、ほぼ不可能なため、各社から自由に利用できる音声認識API(Application Programming Interface)が提供されています。

- Apple Siri API
- Google Cloud Speech API
- IBM Watson Speech to Text API
- Microsoft Bing Speech API　　など

これらを利用することにより各社が独自に開発した音声認識を容易に取り入れたプログラム開発を行うことができます。しかし、ユーザー登録、無料利用枠に限りがある、インターネット接続環境が必要など、いくつか授業で利用するには制約があります。講師側の立場として、学生全員のアカウント管理などは容易なことではありません。そこで、この制約をほぼすべてなしで利用することができる音声認識 Web Speech API に注目しました。

この Web Speech API は、W3C(World Wide Web Consortium) においてブラウザ上で実行できる音声認識として開発グループが存在しています。

URL : https://www.w3.org/community/speech-api/

その開発が行われ、仕様などが公開されています。

URL : https://wicg.github.io/speech-api/

URL : https://github.com/WICG/speech-api

URL : https://developer.mozilla.org/ja/docs/Web/API/Web_Speech_API

Web Speech API のデモが、Google.com で公開されているので、試してみてもよいでしょう。ただし、Google Chrome(もしくは互換ブラウザ)を利用してアクセスしてください。

URL ： https://www.google.com/intl/ja/chrome/demos/speech.html

1.2　教材の概要

Web Speech API 利用に際して、以下のような特徴を持っています。これらを踏まえてシステムを検討しました。

- ブラウザがあれば実行可能(スマートフォンで問題ない)
- 開発は JavaScript
- 一部は、オフラインでも実行可能

この教材で利用した機器について、以下に示します。

- アクセスポイント(Buffalo 社製 ： WAPS-AG300H)
- 操作端末(ASUS 社製タブレット ： Zen Pad 8)
- パソコン(Windows 10 Enterprise) ＋ 仮想化環境(Hyper-V)
- Web サーバー(Linux ： CentOS8 ＋ Apache)
- ハードウェア制御(Espressif Systems 社製 ： ESP32-DevKitC)
- ブレッドボード(大きいサイズ)、LED、リレー、サーボモーターなどの制御対象

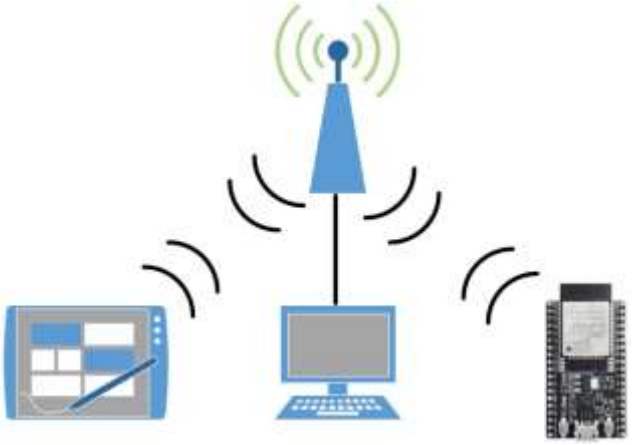
ハードウェア制御には、Espressif Systems 社の ESP32-DevKitC(以下、ESP32) を採用しました。採用した理由は、Arduino と同一の開発環境を持ち、通信機能があるためです。Arduino の利用は、1年生時の授業において利用経験があるため、プログラム開発が行いやすいという利点がありました。

またこの教材を活用する上で、唯一変更できないものは、 ESP32 のみです。この教材にあるプログラムは、この機器に合わせて作成しています。ただし、 ESP32 の互換ボードであれば、補償はできませんが動作するはずです。その他については、実際の実習環境に合わせ自由に変更可能です。

このシステムでは、Hyper-V 仮想環境を利用しています。Windows 10 Pro、Enterprise Edition において利用可能なため、Windows 10 Home Edition で開発を行うようであれば、Oracle VirtualBox を利用することで、同様の環境を構築することができます。また、Docker を利用したコンテナ型仮想環境を利用しても開発は可能です。

教材での操作端末には、Android タブレットを利用しています。スマートフォン、有線 LAN で接続したパソコンでも動作します。Google Chrome 系ブラウザが動作すれば問題ありません。

以上を踏まえ構成する開発システムを、以下のようにしました。



アクセスポイントを中心としたネットワーク構成とし、操作用タブレット、ホームページ配信用 Web サーバー（パソコン）、ハードウェア制御用マイコン（ESP32）で実習を行います。

教材の内容を、以下に示します。

- ESP32
  - ESP32 の開発環境を整える
  - ESP32 を利用したプログラム開発1（LED 制御）
  - ESP32 を利用したプログラム開発2（リレー制御）
  - ESP32 を利用したプログラム開発3（モーター制御）
  - ESP32 を利用したプログラム開発4（無線 LAN の子機側として制御）

- パソコン、仮想化環境
  - Hyper-V の導入
  - Hyper-V に Linux をインストール、環境設定
  - Web サーバーの構築、環境設定
  - SSL サーバーの構築、環境設定
  - PHP の導入、Web サーバーとの連携

- ホームページ作成、API 利用
  - HTML、JavaScript について
  - PHP について
  - Web Speech API について

- ESP32 音声制御
  - 動作確認
  - 全体連携

2　ESP32 の利用

今回は、ESP32 を利用してハードウェア制御を行います。また、無線 LAN 機能を活用していきます。

ESP32 は、Arduino 互換でありながら、無線 LAN、Bluetooth などの通信機能を搭載した製品です。Espressif Systems 社が製作した SoC（System on Chip）機器です。SoC とは、すべての機能を1つのチップ（CPU）に搭載したものです。ESP32 には、下位製品として ESP8266 というものもあります。以下に比較性能を示します。

表 1：主な種類と仕様 ※2

Model	Arduino Nano	ESP8266	ESP32
CPU	8bit、16MHz シングルコア	32bit、80MHz シングルコア	32bit、240MHz デュアルコア
RAM	2KB	96KB	520KB
メモリ	32KB	2MB	4MB
無線 LAN	-	802.11b/g/n	802.11b/g/n
Bluetooth	-	-	4.2 BR/EDR、BLE
GPIO	14pin	17pin	32pin

今回の実習のように無線 LAN 機能だけでよければ、ESP8266 を利用して構いません。

ESP32 を利用してプログラム開発を行う際、以下の 3 種類が用意されています。

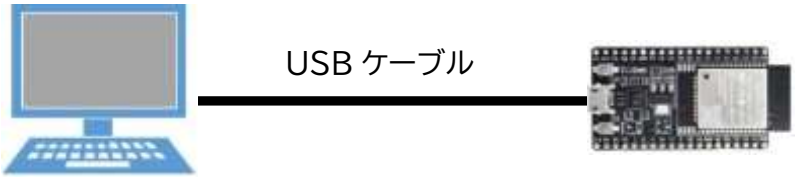
- ESP-IDF
- Micro Python
- Arduino IDE

こちらも今回の実習では、どの開発環境を利用しても行うことができます。

Arduino IDE は、1年生時に操作したこともあり、使い方にも慣れているはずですので、こちらを利用してプログラム開発を行います。

2.1　ESP32 を利用するための Arduino IDE の環境設定

Arduino のプログラム開発と同じく USB ケーブルでパソコンと ESP32 を接続し、パソコン上で作成したプログラムを ESP32 に書き込み、動作確認を行います。制御構文などの基本構文は、そのまま利用することができます。ただし、ESP32 独特の記述が一部あり、機能を利用するためには、Arduino とは異なる命令を利用することになります。



ESP32 を Arduino IDE で利用するためには、Arduino IDE にボード情報を追加する必要があります。

そのため、追加のボードマネージャ URL を指定する必要があります。

Arduino IDE を起動させ、

ファイル ＞ 環境設定 を開き、追加のボードマネージャの URL を記述します。

追加する URL 情報は、タイミングにより変更されています。以下の URL から確認してください。

https://github.com/espressif/arduino-esp32/blob/master/docs/arduino-ide/boards_manager.md

環境設定後、利用するボード情報をダウンロードします

ツール ＞ ボード ＞ ボードマネージャ を開き、ESP を検索し、esp32 をインストールします。

ここまで出来ると、ボードを選び、プログラムを記述し、ESP32 に書き込むことになります。

ボード選択は、ツール ＞ ボード ＞ ESP32 Dev Module を選択します。

適切なシリアルポートを選択し、ボード情報の取得ができれば問題ありません。

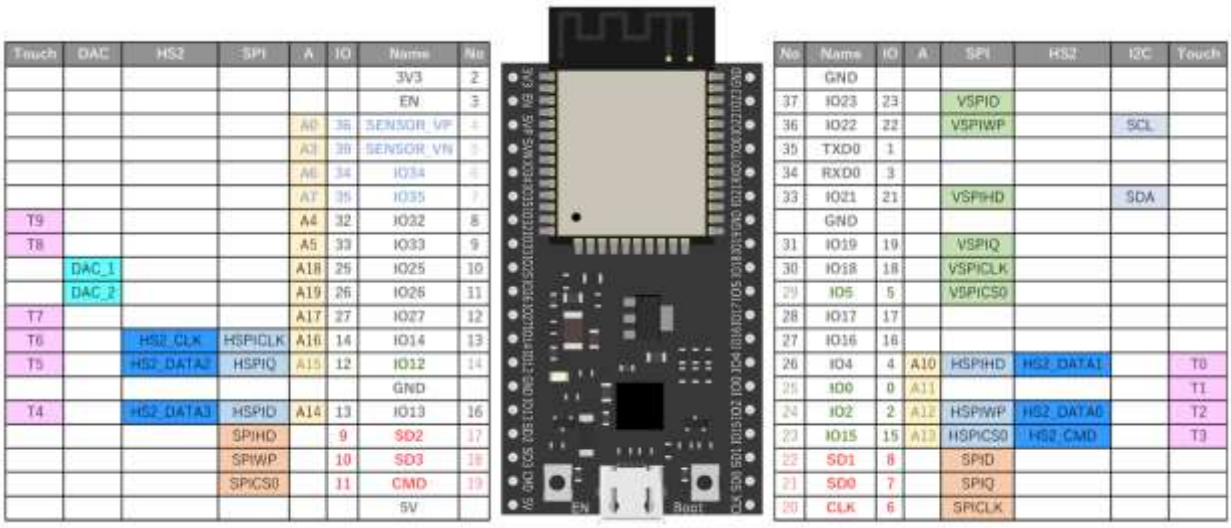
2.2　ESP32 のピン配置、機能

ESP32 を利用して、ハードウェア制御を行うために、ピン配置を確認します。具体的な内容は、データシートで確認できます。

https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

基本的な制御方法も含め、以下の URL のサイトが参考になります。

URL : https://ht-deko.com/arduino/esp-wroom-32.html



Touch	DAC	HS2	SPI	A	IO	Name	No
						3V3	2
						EN	3
				A0	36	SENSOR_VP	4
				A3	39	SENSOR_VN	5
				A6	34	IO34	6
				A7	35	IO35	7
T9				A4	32	IO32	8
T8				A5	33	IO33	9
	DAC_1			A18	25	IO25	10
	DAC_2			A19	26	IO26	11
T7				A17	27	IO27	12
T6		HS2_CLK	HSPICLK	A16	14	IO14	13
T5		HS2_DATA2	HSPIQ	A15	12	IO12	14
						GND	
T4		HS2_DATA3	HSPID	A14	13	IO13	16
			SPIHD		9	SD2	17
			SPIWP		10	SD3	18
			SPICS0		11	CMD	19
						5V	

No	Name	IO	A	SPI	HS2	I2C	Touch
	GND						
37	IO23	23		VSPID			
36	IO22	22		VSPIWP		SCL	
35	TXD0	1					
34	RXD0	3					
33	IO21	21		VSPIHD		SDA	
	GND						
31	IO19	19		VSPIQ			
30	IO18	18		VSPICLK			
29	IO5	5		VSPICS0			
28	IO17	17					
27	IO16	16					
26	IO4	4	A10	HSPIHD	HS2_DATA1		T0
25	IO0	0	A11				T1
24	IO2	2	A12	HSPIWP	HS2_DATA0		T2
23	IO15	15	A13	HSPICS0	HS2_CMD		T3
22	SD1	8		SPID			
21	SD0	7		SPIQ			
20	CLK	6		SPICLK			

図 2：ESP32 のピン配置 ※3

今回の実習でのハードウェア制御に必要な個所は、2つです。

- IO のピン番号
- アナログ出力のピン番号

この実習が終わった後で、I2C、SPI などの利用も想定されるようであれば、確認してください。

ESP32 では、IO においていくつか制約があり、利用しにくい IO が存在しています。

- IO0～3 システム用として利用されている
- IO6～11 は、メモリ制御、SPI と接続しているため併用不可
- IO34～39 は、出力として利用不可（入力専用）

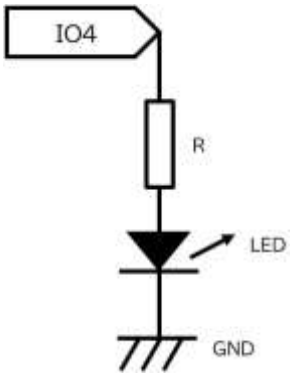
この点を踏まえ、配線を行います。

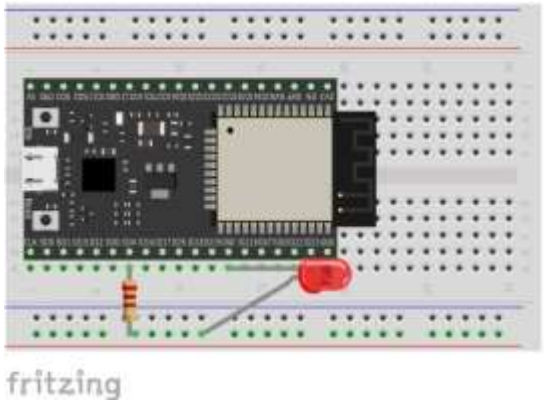
また、Arduino の動作電圧が 5V であるのに対し、ESP32 は 3.3V で動作します。そのため利用する負荷などに注意が必要です。

2.3 LED 点灯デジタル制御

デジタル制御に関しては、Arduino の書き方と違いはありません。

IO4 を利用し、LED を点滅させます。ブレッドボードの配線図を参考に、配線を行ってください。





以下のプログラムを書き込み、動作確認を行います。

```
const int LEDPin = 4;

void setup() {
  pinMode(LEDPin, OUTPUT);
}
void loop() {
  digitalWrite(LEDPin, HIGH);
  delay(250);
  digitalWrite(LEDPin, LOW);
  delay(250);
}
```

● 練習問題

IO5 にも LED を接続し、IO4、IO5 が 0.5s 毎に交互に点灯するプログラムを作成しなさい。

2.4　LED 点灯 PWM 制御

デジタル制御に関しては、Arduino の記述上変更点はありません。しかし、アナログ関連制御については、違いがあります。記述方法を理解する必要があります。

PWM 制御に関する細かい内容については省略しますが、PWM 制御を利用することができれば、モーター制御にも活用できます。

ESP32 は、PWM 出力を 16 個(チャンネル)同時に出力することができます。このチャンネルをアナログピンに指定することで、利用可能となります。チャンネルは、0～15 です。

PWM 制御には、以下の 3 つの命令が必要になります。

- ledcSetup(チャンネル番号, 周波数, ビット数)
  : 利用するチャンネル、PWM 動作指定
- ledcAttachPin(ピン番号, チャンネル番号)
  : チャンネルを割り当てるアナログピン指定
- ledcWrite(チャンネル番号, 値)　: 値に応じた PWM 制御

アナログピン 10 番を利用して、PWM 制御を行います。前のページのピン配置から確認するとアナログピン 10 番は、IO4 番となります。

以下のプログラムを書き込み、動作確認を行います。

```
void setup() {
  // チャンネル 0、周波数 5000Hz、8bit(= 256 段階制御)
  ledcSetup(0, 5000, 8);
  ledcAttachPin(A10, 0);          // アナログピン A10 を チャンネル 0 を指定
  ledcWrite(0, 0);                // 消灯状態に設定
}
void loop() {
  for (int i=0; i<=255; i+=5) {   // 徐々に明るく
    ledcWrite(0, i);              // チャンネル 0(=A10) を設定
    delay(250);
  }
  for (int i=255; i>=0; i-=5) {   // 徐々に暗く
    ledcWrite(0, i);
    delay(250);
  }
}
```

● 練習問題

アナログピン 5 番に LED を接続し、0.1s 毎に徐々に明るくなるプログラムを作成しなさい。

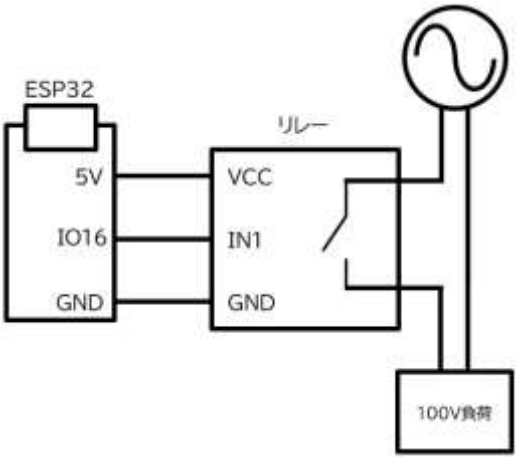
2.5　リレー制御

リレー制御ができれば、100V 機器の電源操作が可能になります。IoT においてよく利用されている制御です。制御方法は、デジタル制御です。

制御対象は、何でも構いませんが 100V 側の配線には、十分に注意して行ってください。100V の配線が危険であれば、12V、24V の負荷でも構いません。

以下は、今回利用したリレーです。同じものである必要はなく同等の機能を有していれば問題ありません。

VCC に5V、GND に GND の配線を行います。IN1 を IO16 と接続しています。



以下のプログラムを書き込み、動作確認を行います。

```
const int relayPin = 16;

void setup() {
  pinMode(relayPin, OUTPUT);
}
void loop() {
  digitalWrite(relayPin, HIGH);
  delay(1000);
  digitalWrite(relayPin, LOW);
  delay(1000);
}
```

ESP32

2.6　サーボモーター制御

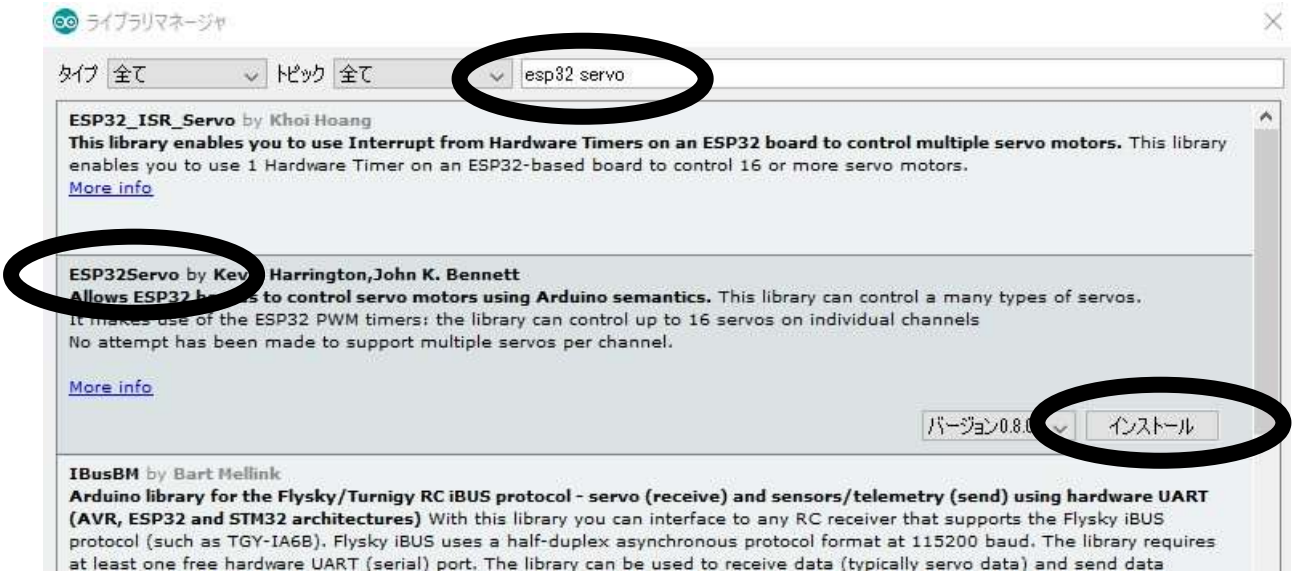
サーボモーター制御ができると、家庭用タンブラスイッチをオンオフするなどが可能になります。

PWM を利用すると制御可能ですが、サーボモーター制御用の専用ライブラリが用意されているので、こちらを利用して制御します。

ライブラリを追加します。

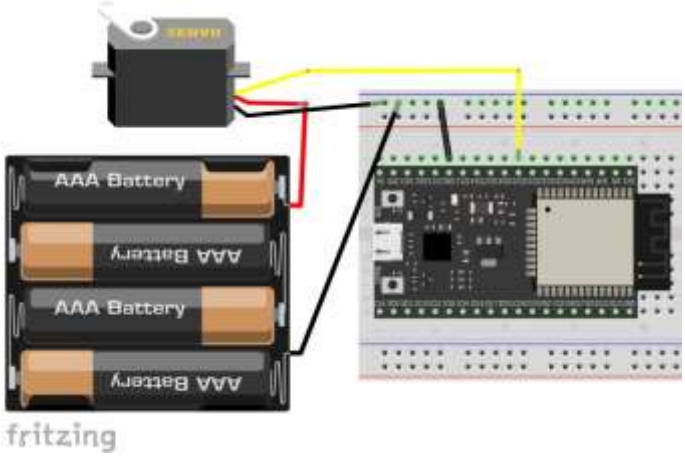
スケッチ ＞ ライブラリをインクルード ＞ ライブラリを管理... から、 esp32 servo を検索

ESP32Servo ライブラリをインストールします。



サーボモーターには、TowerPro MG995 を利用しました。これも同じものである必要はなく同等の機能を有していれば問題ありません。

ESP32 の IO25 から出力したシグナルをサーボモーターに送ります。



本来であれば、ESP32 の信号出力は 3.3V で動作しており、サーボモーターへの信号入力が 5V でなければならないことがあります。場合によっては、信号が入らない可能性があるため 電圧レベル変換器 を利用した方がよいかもしれません。今回は、動作が行えたため利用しませんでした。

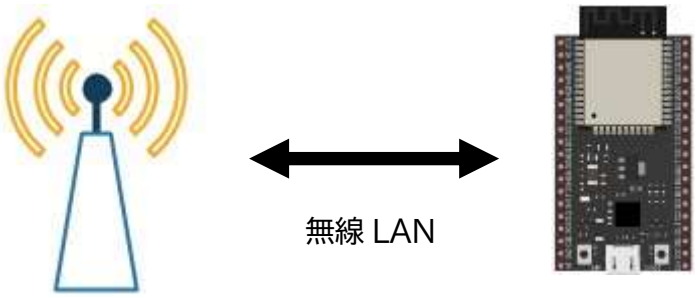
以下のプログラムを書き込み、動作確認を行います。

```
#include <ESP32Servo.h>          // ライブラリ利用宣言

// サーボモーター利用宣言
Servo servoDo;
int servoPin = 25;

void setup() {                           // サーボモーター利用初期設定
  servoDo.setPeriodHertz(50);   // 50Hz
  servoDo.attach(servoPin);       // 信号を送るピン番号指定
  servoDo.write(90);                  // 初期位置(90 度)を指定
}
void loop() {
  servoDo.write(150);      // 150 度の位置
  delay(1000);
  servoDo.write(30);        // 30 度の位置
  delay(1000);
}
```

2.7　無線 LAN 子機側として設定

今回のネットワーク構成では、ESP32 を無線 LAN の子機として利用します。動作に関しては、アクセスポイントから DHCP を利用した自動で IP アドレスを割り振る形は、制御がやりにくくなるため固定 IP アドレスを利用することとしました。

ESP32 は、2.4GHz帯の無線 LAN にしか対応していないため、動作させるアクセスポイントの無線 LAN の電波を設定してください。



アクセスポイント（2.4GHz）　　固定 IP アドレス設定

アクセスポイント設定は、以下のようにします。

SSID	speechapidemo
パスワード（WPA2）	polytechtest

ネットワーク体系は、以下のようにします。サブネットマスクは、24bit です。

	IP アドレス
アクセスポイント	192.168.41.1
ESP32	192.168.41.32

今回の実習では、ESP32 をインターネットに接続する必要はありません。また、ルーティングを行う必要がないため同一ネットワークで展開します。

状況に合わせて、ネットワーク体系は、変更してください。

プログラムは、サンプルプログラムを利用すると比較的楽に設定が可能です。

ファイル ＞ スケッチ例 ＞ WiFi ＞ WiFiClientStaticIP を参考に、一部を変更します。

以下のプログラムを書き込み、動作確認を行います。

```
#include <WiFi.h>

// 接続するアクセスポイント指定
// 注意点として、ESP は 2.4GHz(11g 、11n)しか認識できない
const char* ssid          = "speechapidemo";
const char* password      = "polytechtest";

// 固定 IP 設定
IPAddress local_IP(192, 168, 41, 32);    // IP アドレス
IPAddress gateway(192, 168, 41, 1);      // ゲートウェイ(今回はなくても)
IPAddress subnet(255, 255, 255, 0);      // サブネットマスク
IPAddress primaryDNS(8, 8, 8, 8);        // DNS1 optional(今回はなくても)
IPAddress secondaryDNS(8, 8, 4, 4);      // DNS2 optional(今回はなくても)

void setup() {
  Serial.begin(115200);

  // IP アドレス関連設定
  if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
    Serial.println("STA Failed to configure");
  }

  // アクセスポイント接続
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // 設定状況確認
  Serial.println("");
  Serial.println("WiFi connected!");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  Serial.print("ESP Mac Address: ");
  Serial.println(WiFi.macAddress());
  Serial.print("Subnet Mask: ");
  Serial.println(WiFi.subnetMask());
  Serial.print("Gateway IP: ");
  Serial.println(WiFi.gatewayIP());
  Serial.print("DNS: ");
  Serial.println(WiFi.dnsIP());

void loop() {
}
```

アクセスポイント接続が出来たうえで、先ほどまで見てきたハードウェア制御を組み合わせていくことになります。

3　Hyper-V 環境構築

Web Speech API を動作させるためには、API を実行するブラウザが動作しているパソコン上で Web サーバー構築しても動かせません。そのため、Web サーバーを別のパソコンで動作させる必要があります。さらに、SSL サーバー構築し、Web サーバーと連携させる必要があります。

ブラウザ ＋ Web サーバー
＋ SSL サーバー

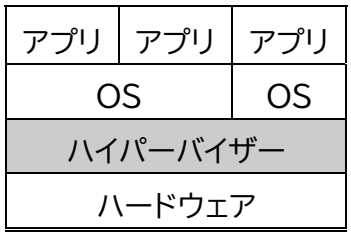
ブラウザ　　Web サーバー ＋ SSL サーバー

パソコンを2台利用しなければ、動作試験をすることができないことになります。そこで、ハードウェアを仮想化できる仮想環境を利用します。

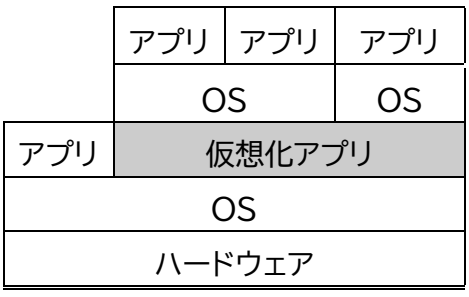
仮想環境を構築するには、いくつか方法があります。代表的には、以下の通りです。

- ハイパーバイザー型
- ホスト型
- コンテナ型

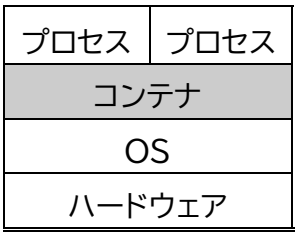
ハイパーバイザー型は、実ハードウェア上に仮想環境を構築するものです。構築するためには、OS 再インストールなどが必要になるはずです。ホスト型は、動作中の OS に仮想化アプリケーションをインストールすることで動作させることが可能です。コンテナ型は、ホスト型に近いですが OS をインストールせずハードウェア資源をホスト OS と共有し、動作が軽いところがメリットです。



ハイパーバイザー型



ホスト型



コンテナ型

共通点は、ハードウェア(CPU、メモリ容量など)が十分に搭載されている必要があります。

3.1 Hyper-V のインストール、環境構築

仮想環境の構築については、どれを採用してもこの実習には問題はありません。今回は、ホスト型の Hyper-V を利用し仮想環境を構築します。Windows 10 Professional、Enterprise Edition であれば、無償で導入できます。

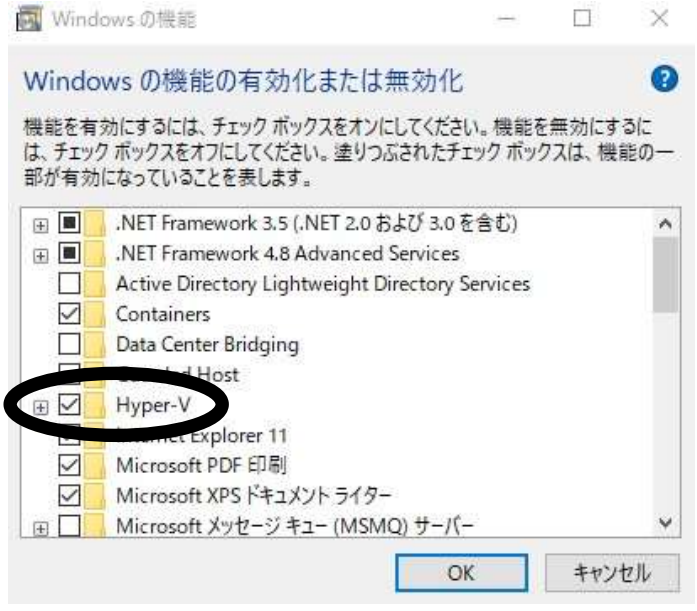
<table>
<tr><td></td><td colspan="2">Web サーバー</td><td>SSL サーバー</td></tr>
<tr><td></td><td colspan="3">ゲスト OS（Linux）</td></tr>
<tr><td>アプリ</td><td colspan="3">Hyper-V</td></tr>
<tr><td colspan="4">ホスト OS（Windows）</td></tr>
<tr><td colspan="4">ハードウェア</td></tr>
</table>

ホスト型

もし、Windows 10 Home Edition を利用するのであれば、Oracle VirtualBox を利用することを検討してください。機能的な、違いはありません。

仮想環境を動作させるためには、UEFI/BIOS の設定を利用して仮想化支援技術（Virtualization Technology）を有効にする必要があります。パソコンを再起動させ、確認します（マザーボードにより表示が異なります。取扱説明書などで確認してください）。

Hyper-V のインストール は、コントロールパネルから行います。
コントロールパネル ＞ プログラム ＞ Windows の機能の有効化または無効化
Hyper-V にチェック ＞ OK をクリック



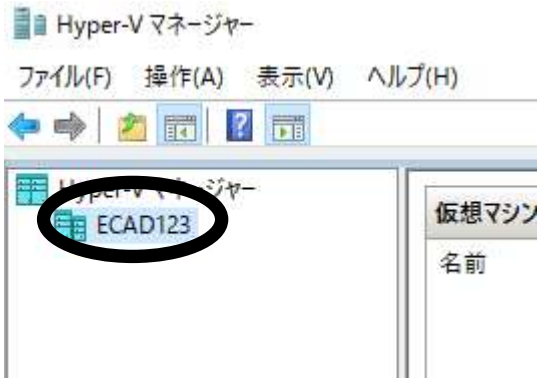
再起動後に、Hyper-V が利用できるようになります。

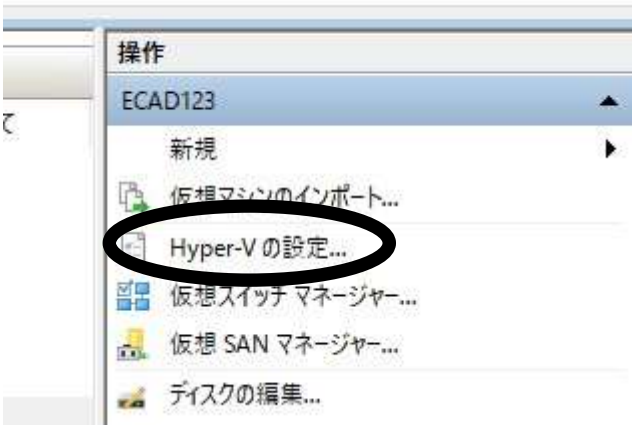
Hyper-V 環境構築

Hyper-V をインストールすると、Hyper-V マネージャー が利用できます。
スタートの Windows 管理ツール ＞ Hyper-V マネージャー を起動させます。

Hyper-V の管理操作は、上記の右側画面から行います（他にもコマンド操作ができるようであれば、PowerShell を利用して管理操作を行うことも可能です）。

まず、基本的な設定から行っていきます。
Hyper-V マネージャー の左側に操作中のコンピューター名が記載されます。これを選択します。下の例では、コンピューター名 ECAD123 ということになります。
画面右側の Hyper-V の設定 を選択します。





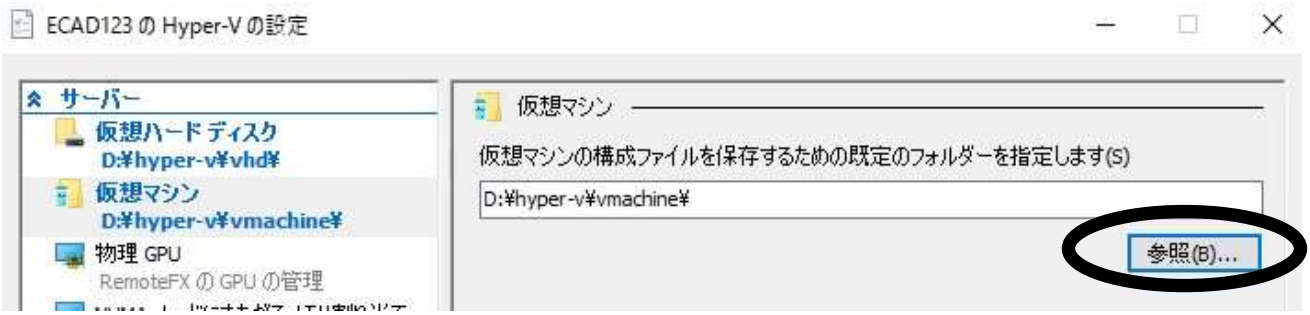
以下のような、設定画面が現れます。

基本的にあまり変更する必要ありません。ただし、仮想マシンをたくさん構築すると実ハードディスク、または SSD の容量を大量に必要とします。もし、C ドライブ以外のドライブに仮想マシンのハードディスク領域が移動できるとホスト OS である Windows の動作に影響を与えずに済みます。

今回は、D ドライブに仮想ハードディスク、仮想マシンを移動させるようにします（必ずしもこの作業はしなければならないものではありません）。

以下のように、指定します（フォルダがなければ作成します）。

仮想ハードディスク	D:¥hyper-v¥vhd
仮想マシン	D:¥hyper-v¥vmachine



設定できれば、適用します。

3.2　仮想マシン環境作成

仮想マシン環境を新規作成します。作成方法が、2つあります。

- クイック作成
- 通常の新規作成

作成方法としての クイック作成　では、簡単に作成できます（大きなイメージのダウンロードを行うため、高速なインターネット接続環境が必要です）。

以下のような画面が表示され、作成したい仮想マシンを選ぶだけです。

このクイック作成をみんなで選んで実行してしまうと回線がパンクしてしまいます。そのため、今回、こちらは利用しません。

Hyper-V 環境構築

今回は、仮想マシン環境を新規作成します。

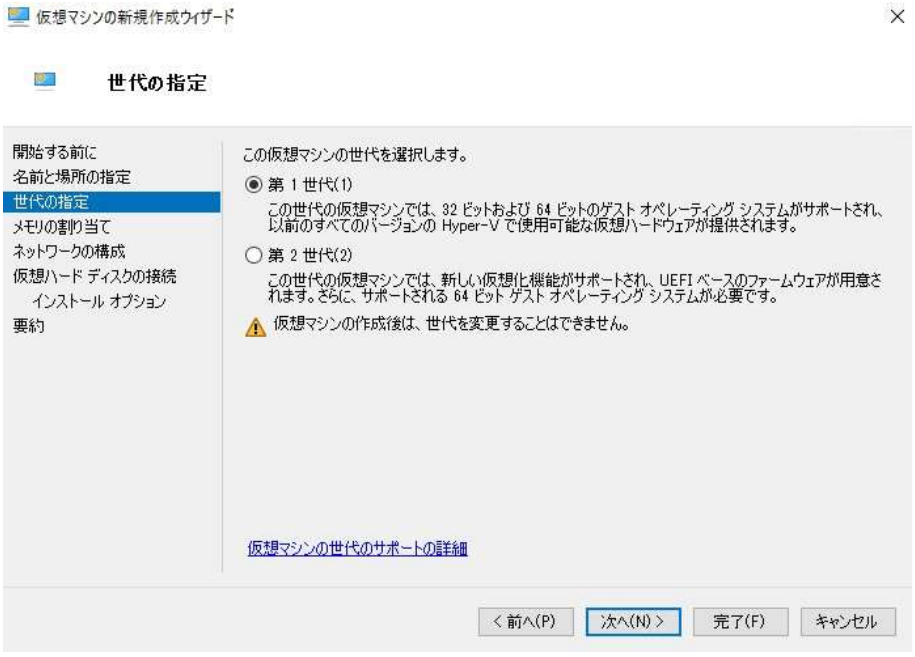
新規作成する場合には、事前にインターネットなどから インストールする OS のイメージディスクファイル(iso ファイル)を事前に用意します。

今回は、CentOS8 を用意しました。Hyper-V マネージャー から、 新規 → 仮想マシン を選択します。

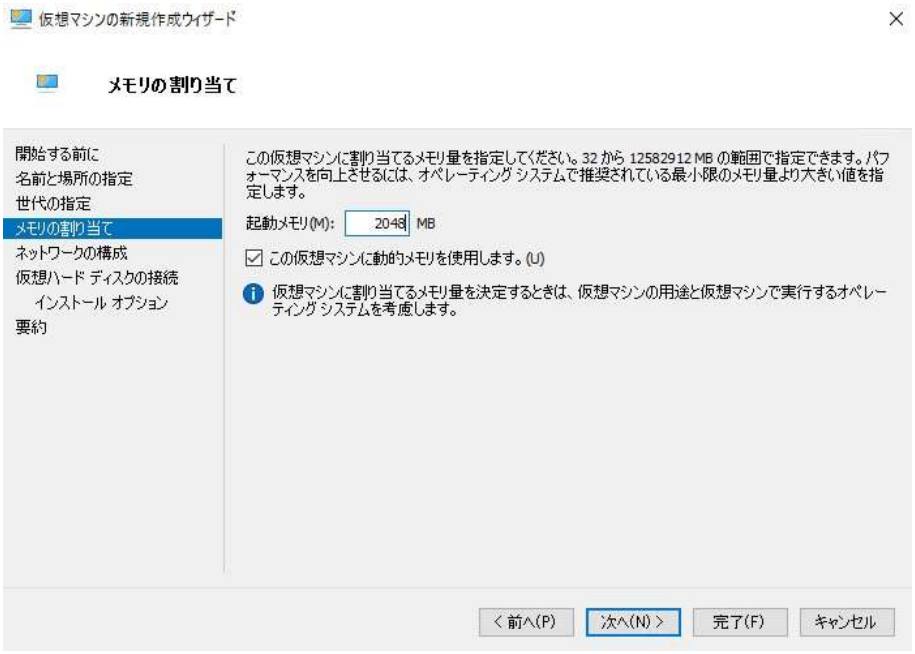
仮想マシンの環境作成は、難しい操作もほとんどありません。ウィザードに答えていくだけで、完成します。

管理用の名前を指定します。何でも構いませんが、後で作成者自身が識別できる情報にします。
今回は、centos8 にしています。

世代情報を指定します。第1世代を選択します。

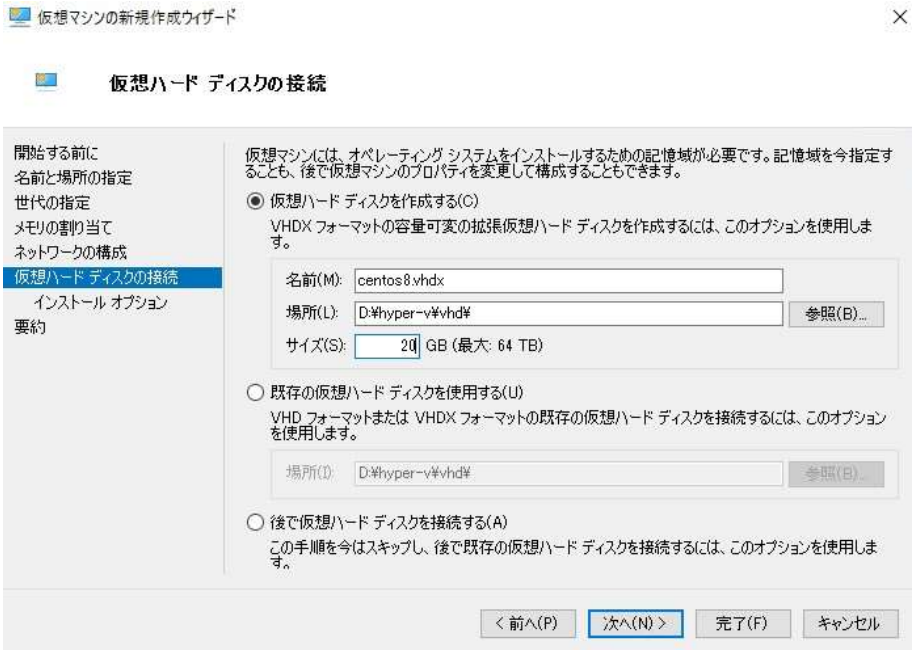


メモリ割り当ては、インストールする OS の仕様に合わせて調整します。インストールする OS のハードウェア要件を確認してください。さらに、ホスト OS(Windows)とのメモリ割合(全メモリ容量)も勘案し、調整します。今回は、あまり大きな作業はしないため 2048MB にしています。

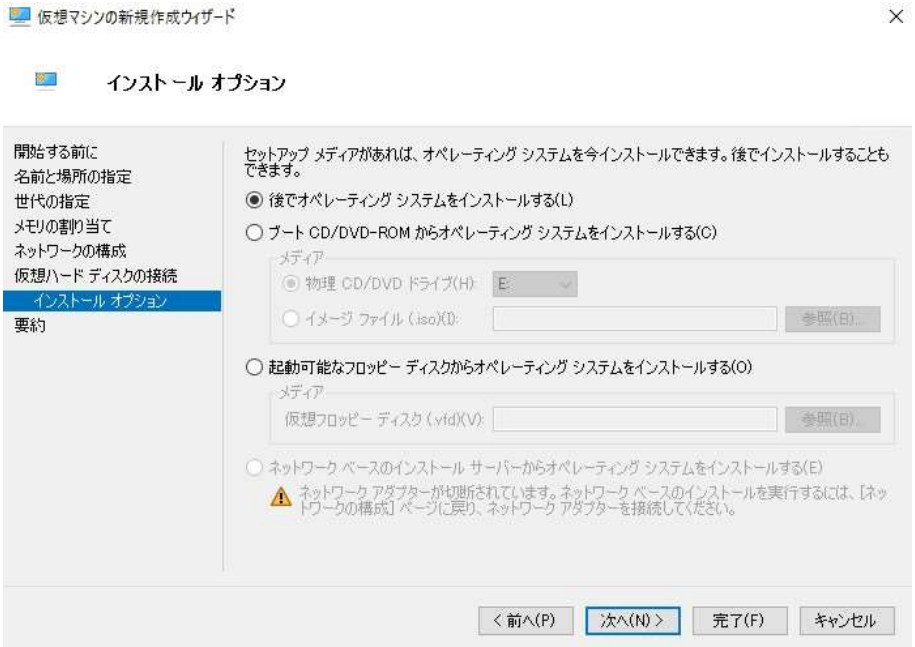


ネットワークの構成は、Hyper-V 設定、仮想マシン設定などができた後にネットワークに接続します。そのため、ここでは接続しないを選択します。

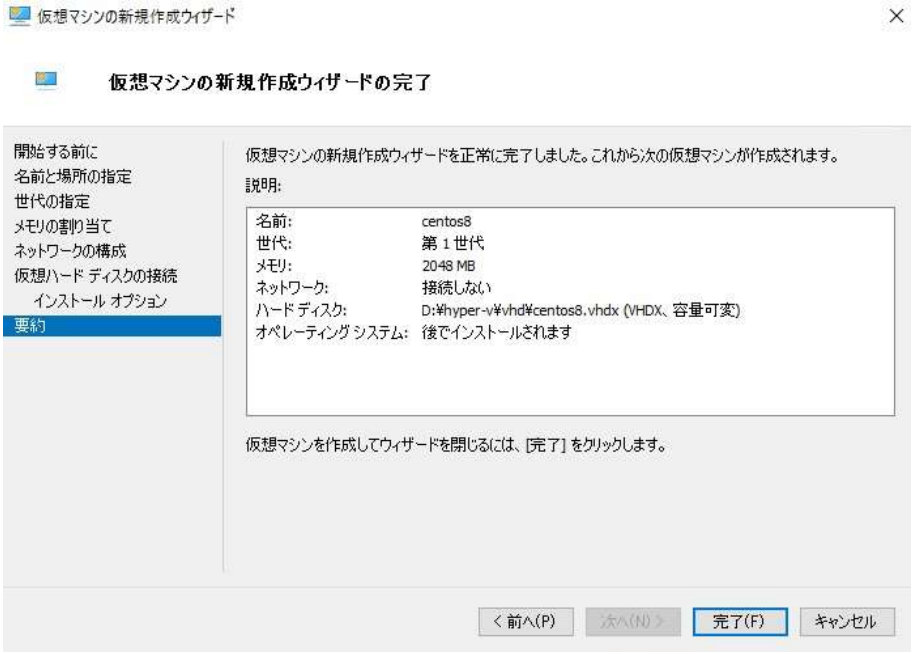
ハードディスク容量もインストールする OS の仕様に合わせます。今回は、20GB にしています。



インストールオプションとして、インストールディスクを指定します。これは、後から設定変更もできます。ここでは、後で OS をインストールする を選択しています。



最後に確認になります。設定に問題があれば、再度、前に戻り調整します。問題なければ、完了を選択します。



Hyper-V マネージャー に、仮想マシンが追加されていることが確認できます。

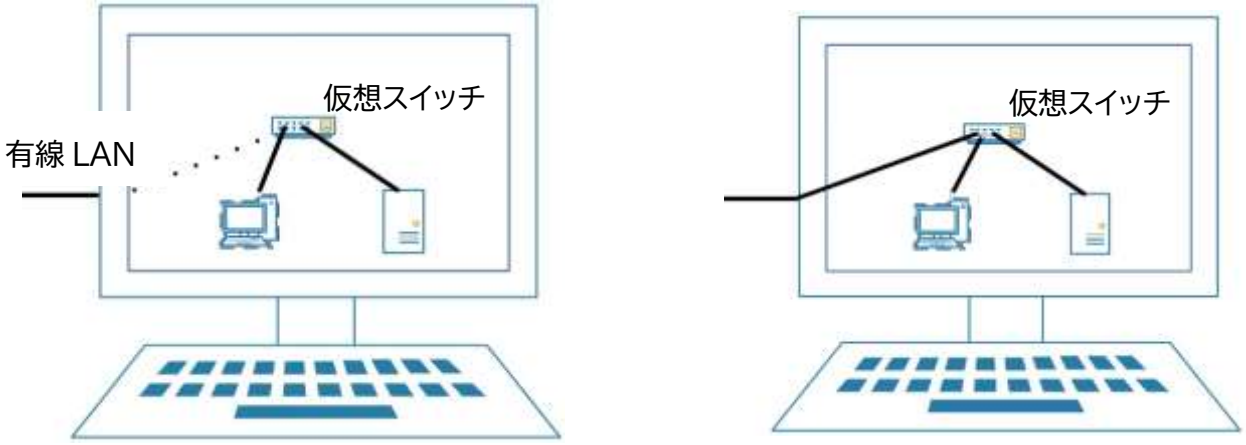
仮想マシンの設定を一部変更します。
作成した仮想マシンを右クリックし、設定 を選択します。

インストールする OS の仕様によりますが、CPU がデュアルコア仕様であれば、プロセッサの数を増やします。さらに、事前に用意したインストールディスクイメージ（iso ファイル）を指定します。

3.3　Hyper-V のネットワーク設定（仮想スイッチの作成）

仮想環境では、仮想ネットワークを構成することになります。仮想ネットワークの設定では、大きく分け 2 つのタイプがあります。

- ホスト OS の外のネットワークにパケット出さない閉じたネットワーク
- ホスト OS と同一のネットワークにパケットを出せるネットワーク



閉じたネットワーク環境は、実験用として適した環境と言えます。現実のネットワークに影響を与えずに作業を行うことができます。

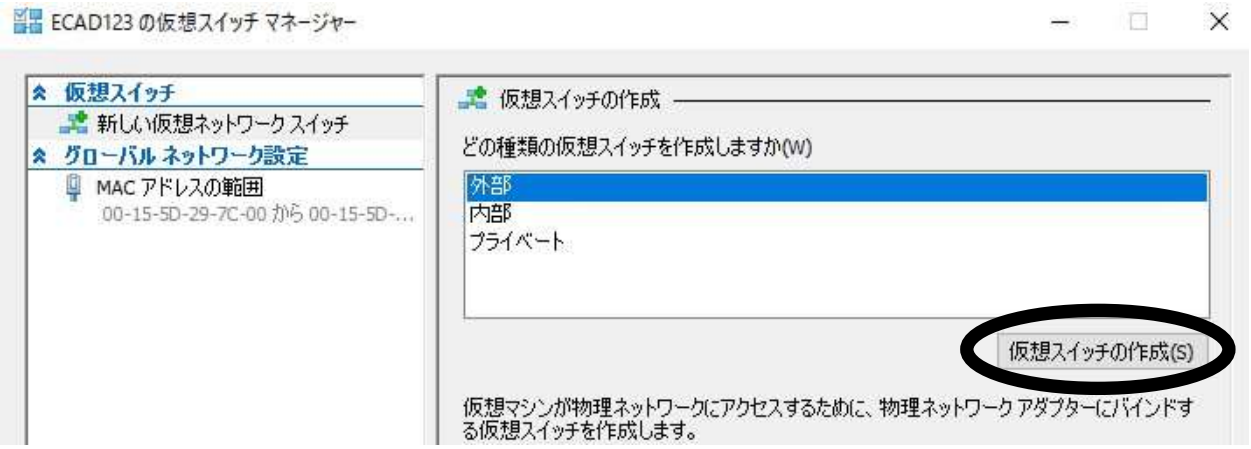
パケットを現実のネットワークに送出できる環境は、現実のネットワークの一部として運用することができます。

これ以外にも仮想ネットワークは存在しますが、概ねこの2つのどちらかを利用することになります。

今回は、仮想サーバーを構築し、現実のネットワークと接続する必要があります。

Hyper-V マネージャー ＞ 仮想化スイッチ マネージャー を選択します。

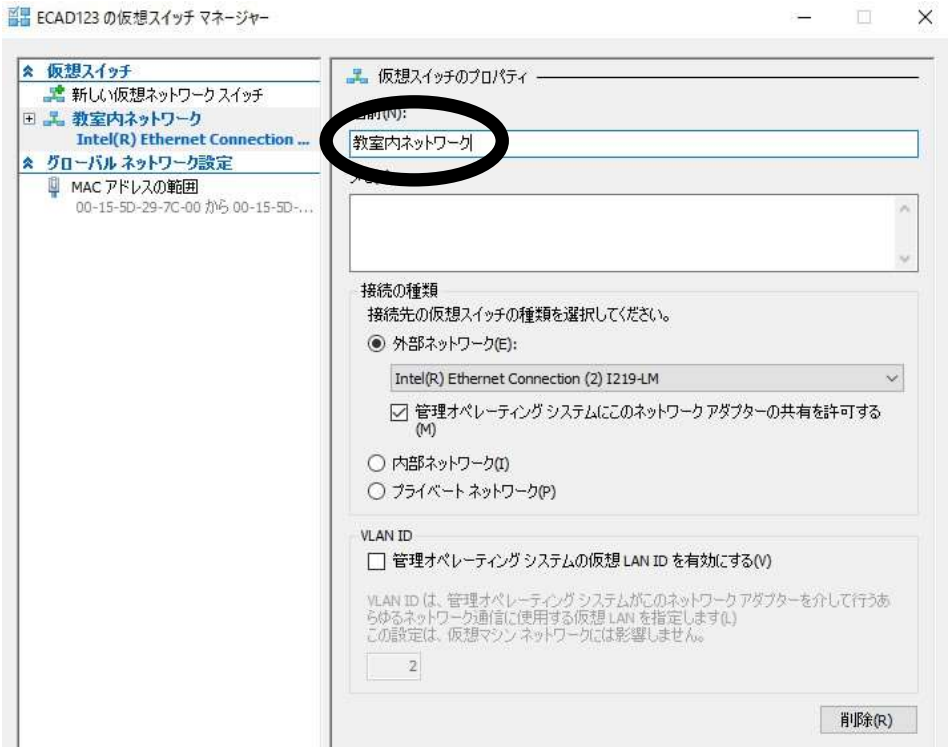
仮想スイッチ マネージャーを選択すると、以下のようなウィンドウが開きます。



ここで外部ネットワークに接続する必要があるため、外部 を選択状態とし、仮想スイッチの作成を選択します。

仮想スイッチには、識別できる名前を付ける必要があります。適当な名前で構いません。

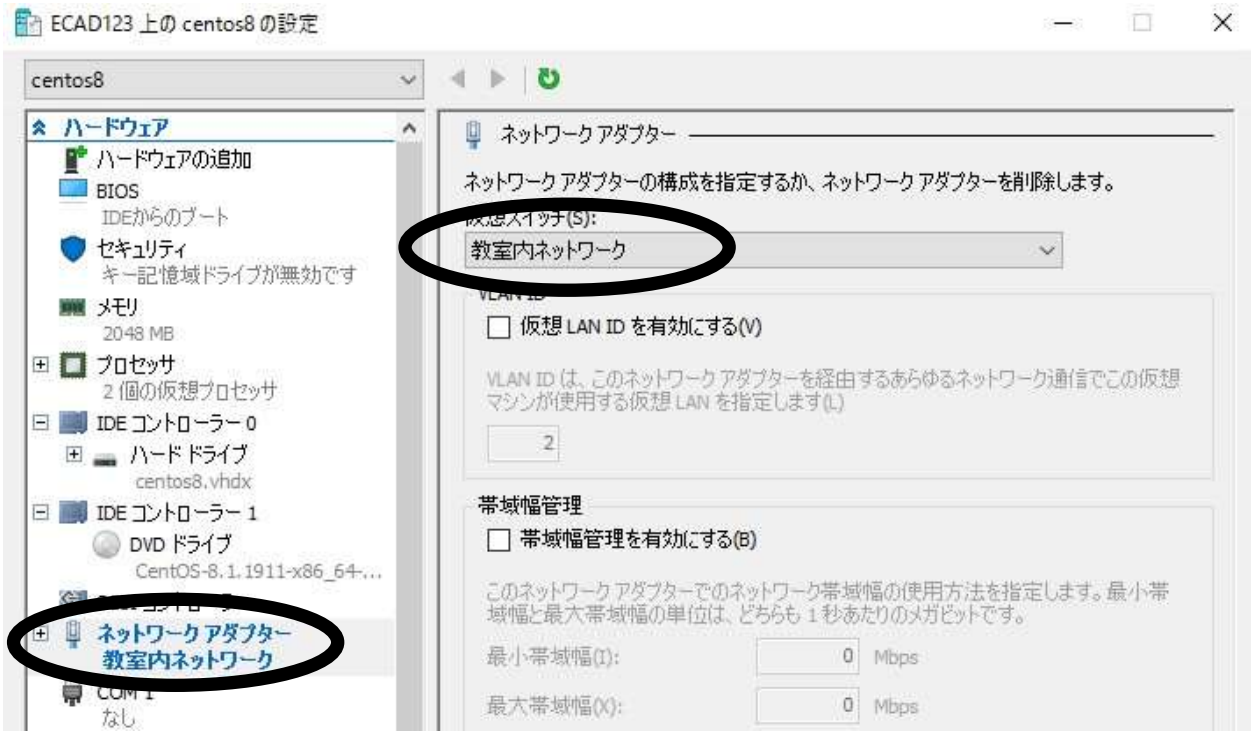
今回は、教室内ネットワーク としています。



3.4　仮想マシンへ仮想スイッチを接続する

仮想スイッチが作成できると、仮想マシンに仮想スイッチを接続します。仮想マシン（centos8）の設定を開きます。

ネットワークアダプター　＞　仮想スイッチ　に先ほど作成した　教室内ネットワーク　仮想スイッチを設定します。



4　CentOS8 のインストール、環境構築

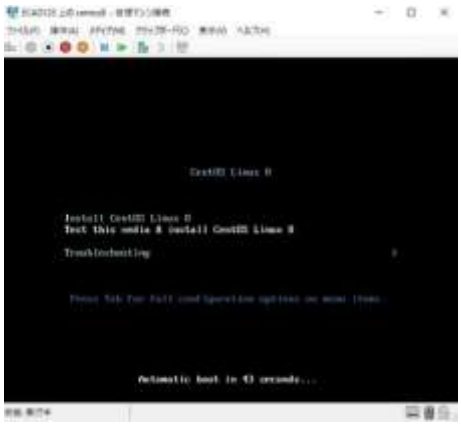
Linux を利用し Web サーバー、SSL サーバーを構築します。これを構築し、操作端末から ESP32 を制御する準備を行います。

今回利用するディストリビューションは、CentOS8 を利用しています。2つのサーバー構築ができれば問題ありません。使い慣れている OS（Debian、Ubuntu など）があればそちらを利用して構いません。

4.1　インストール

Hyper-V マネージャー の仮想マシンから centos8 を選択、接続、起動させます。

OS のインストール自体は、日本語環境を選択していけば、難しいところは特にありません。



途中、タイムゾーンを　アジア東京　にすると後の設定を省略できます（インストール後でも設定変更可能です）。

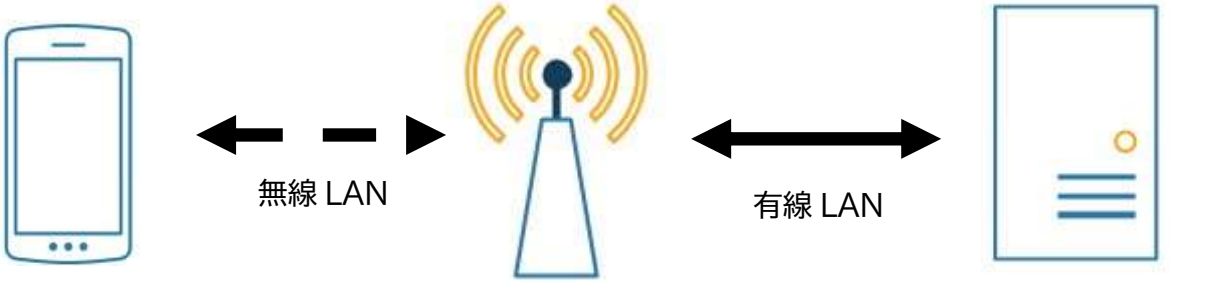
ユーザー、パスワード設定をする必要があります。以下のように設定することとします。

ユーザー名	パスワード
root（管理者）	CentOS87654
testuser（一般）	testpass

インストール終了後、再起動させます。

4.2　ネットワーク設定

今回のネットワーク構成は、以下で確認します。



Web サーバー、SSL サーバー構築後、操作端末から表示を確認します。

アクセスポイント設定は、ESP32 の実習で利用した環境と同じものを利用します。
以下の設定がされていました。操作端末は、事前に接続しておきます。

SSID	speechapidemo
パスワード(WPA2)	polytechtest

ネットワーク体系は、以下のように設定します。サブネットマスクは、24bit です。

	IP アドレス
アクセスポイント	192.168.41.1
ホスト(Windows)	192.168.41.30
CentOS	192.168.41.31
操作端末	192.168.41.100

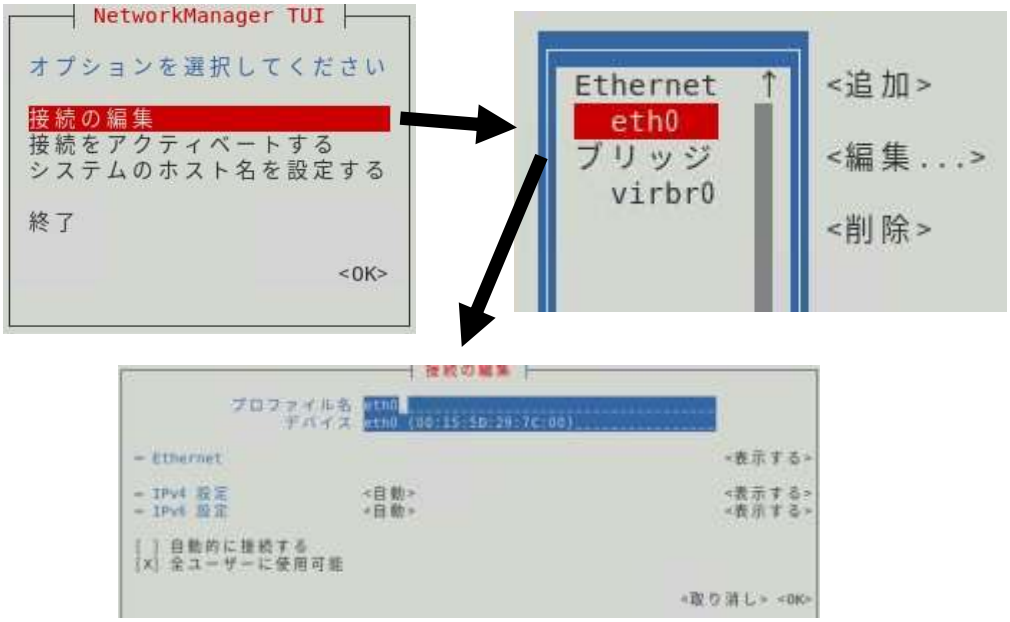
この実習では、ホスト OS、Linux(CentOS)共にインターネットに接続する必要はありません。必要があれば、実習環境において調整してください。
ただし、操作端末だけは API を利用する際、インターネット接続が必要になります。

上記のことを踏まえ、CentOS に IP アドレスなどを設定します。
一般ユーザー(testuser)でログインし、画面左上のアクティビティから、端末を起動させます。

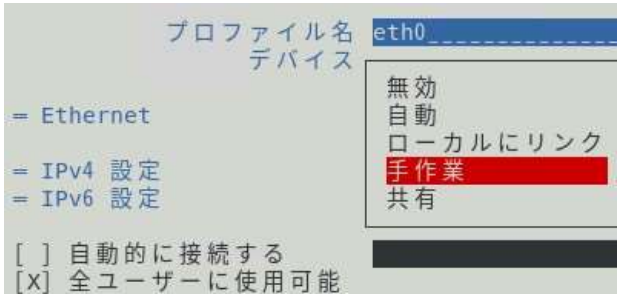
IP アドレス関連の設定を行います。管理者でなければ変更ができません。

```
$ su -                    < 管理者に変更
パスワード:
# nmtui
```

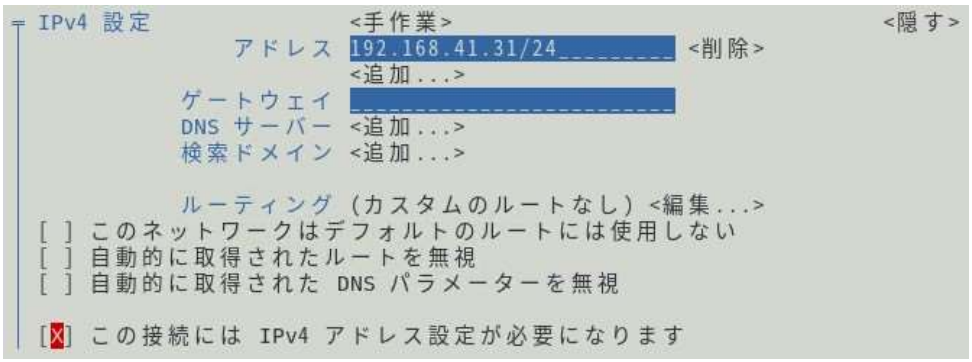
以下のような画面が現れます。該当の Ethernet を選択します(マウス操作はできません)。



上下キーを利用し、IPv4 設定 に移動し、 自動 から 手作業(手動) に変更します。



右に移動し、IPv4 の情報を表示し、事前に決めた設定を行います。

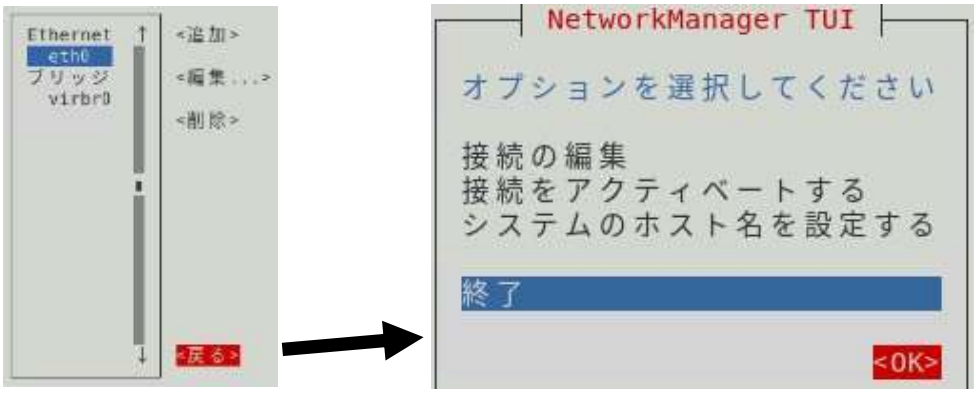


今回の実習では、サーバーはインターネット接続する必要がないためゲートウェイ、DNS などは必要ありません。インターネット環境などが必要であれば、適宜利用環境に合わせて設定します。

今回の実習では、IPv6環境は必要ないので、無効 にします(こちらも環境に合わせて設定します)。

自動的に接続するにもチェックを入れ、右下の <OK> を選択します。

ここまで設定できていれば、右下の <戻る> を選択し、最後に <OK> を選択します。



設定は、このままでは反映されないため NetworkManager を再起動させます。

```
# systemctl restart NetworkManager
```

設定が反映されているかを確認します。

```
# ip add show
```

以下のように、eth0 の inet に IP アドレスの記載があります。

```
[root@localhost ~]# ip add show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:29:7c:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.41.31/24 brd 192.168.41.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
```

ホスト OS(Windows)、アクセスポイント、操作端末に ping を実行し接続確認を行います。

```
# ping 192.168.41.30      < 環境に応じて IP アドレスは、変更します
# ping 192.168.41.1
# ping 192.168.41.100
```

4.3　パッケージの追加インストール環境構築

サーバー構築、その他動作に必要なパッケージの追加インストールをする必要があります。インターネット接続環境があれば問題ありませんが、もしなければインストールイメージ（iso ファイル）を利用することが可能です。

CentOS8 は、事前にパッケージインストールをディスク（イメージファイル）からできる設定が用意されています。これを確認し、設定を行います。以下が設定ファイルになります。

```
$ cat /etc/yum.repo.d/CentOS-Media.repo
内容一部省略
# yum --disablerepo=¥* --enablerepo=c8-media [command]

[c8-media-BaseOS]
name=CentOS-BaseOS-$releasever - Media
baseurl=file:///media/CentOS/BaseOS
```

この記載の中の baseurl に書かれている個所にインストールイメージ（iso ファイル）をマウントすることで利用できます。

やる事が少し多いため管理者（root）になって操作を行います。

```
$ su -
パスワード:
# mkdir /media/CentOS
# mount -t iso9660 /dev/sr0 /media/CentOS/
# ls /media/CentOS/
AppStream  BaseOS  EFI  TRANS.TBL  images  isolinux  media.repo
```

マウントできているようであれば、/etc/yum.repo.d/CentOS-Media.repo 内に記述されている内容を利用すると、簡単に確認することができます。

以下のコマンドを実行し、インストールできるパッケージを確認することができます。

```
# dnf --disablerepo=¥* --enablerepo=c8-media-AppStream list
```

コマンドが長くなるので、alias 設定して構いません（実習に直接関係ないため、内容については割愛します）。

4.4 Web サーバー、SSL サーバーパッケージのインストール

Web Speech API を動作せるためには、Web サーバー、SSL サーバーが必要なためインストールを行います。

今回のインストールする Web サーバーは、Apache。

```
# dnf --disablerepo=¥* --enablerepo=c8-media-AppStream install httpd
```

SSL サーバーは、openSSL。

```
# dnf --disablerepo=¥* --enablerepo=c8-media-AppStream install openssl
```

Apache と openSSL を連携させるために、追加モジュールをインストールします。

```
# dnf --disablerepo=¥* --enablerepo=c8-media-AppStream install mod_ssl
```

これで Web Speech API を動作させためにに必要なパッケージをインストールしたことになります。他にも実習を進めるために必要なものがあれば、インストールしてください。

4.5 PHP のインストール

ESP32 と Web サーバーを連携させる方法は、いくつかあります。その中でも最もシンプルに実現する方法として PHP を採用しました。そのため、Web サーバー（Apache）と PHP が連携できる環境にします。

PHP モジュールのインストールできるリストを確認します。

```
# dnf --disablerepo=¥* --enablerepo=c8-media-AppStream module list php
CentOS-AppStream8 - Media
Name        Stream        Profiles                        Summary
php         7.2 [d]       common [d], devel, minimal      PHP scripting language
php         7.3           common, devel, minimal          PHP scripting language

ヒント: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

環境構築時のデフォルトが PHP 7.2 になっているため、これをインストールします。

```
# dnf --disablerepo=¥* --enablerepo=c8-media-AppStream
                                                module install php:7.2
```

PHP 7.2 がインストールされると、以下のように（表示は、バージョンにより異）なります。

```
# php -v
PHP 7.2.11 (cli) (built: Oct  9 2018 15:09:36) ( NTS )
```

4.6　日本語入力環境の構築 ※4

CentOS8を日本語設定でインストールした後は、日本語表示が可能になっています。しかし、日本語入力ができない状態です。今回の実習では、必ずしも必要ではありませんが、日本語入力環境を整えます。

言語環境を確認します。もし、日本語環境でなければ設定変更を行います(ページ下部参照)。

```
# localectl
System Locale:   LANG=ja_JP.UTF-8
    VC Keymap:   jp
    X11 Layout:   jp
```

日本語入力環境は事前にインストールされていないため必要なパッケージをインストールします。

```
# dnf --disablerepo=¥* --enablerepo=c8-media-AppStream
                                          install libkkc libkkc-data ibus-kkc
```

インストール後は、CentOS 側から通知はありませんが、再起動が必要になります。必ず再起動させます。

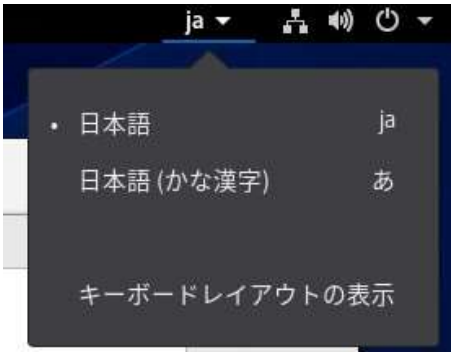
再起動後、マウスを利用し、画面左上の アクティビティ から 設定 を開きます。
左側の項目から、Region & Language を選択します。

入力ソースが、 日本語 になっていることを確認します。日本語以外になっていれば、左下の＋を選択し、日本語 を追加します。

日本語入力機能を追加するには、同様に左下の ＋ を選択し、日本語（かな漢字） を追加します。

追加された結果、以下のようになります。

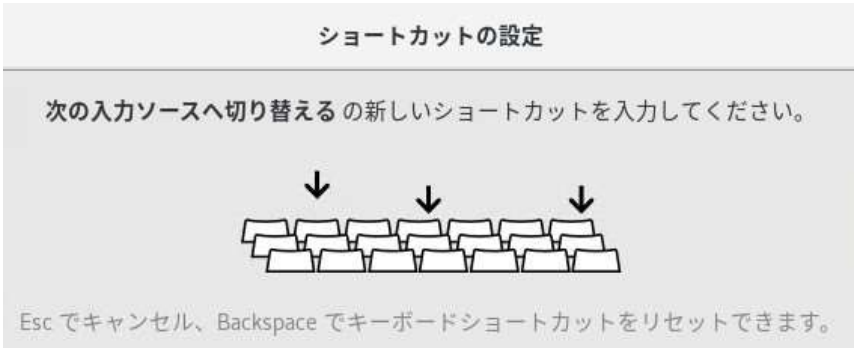
入力ソース に日本語（かな漢字）が追加されると画面右上に変化が現れ、日本語入力に変更できるようになります。



日本語入力への切り替えが、Hyper-V を利用するとやりにくいため、Windows でも利用している 半角/全角 キーを用いて切り替えができるように変更します。

設定から、デバイス ＞ Keyboard ＞ 次の入力ソースへ切り替える を選択します。

その後、以下のウィンドウが表示されます。このタイミングで、 半角/全角 キーを押し、確定します。



タイピングの項目が以下のように変更されます。これで、設定は終了です。

半角/全角 キーを押すことで入力モードを変更できるようになります。

4.7　Web サーバー(Apache)の設定

Web サーバーを動作させます。設定自体は、Web ページが操作端末から確認できれば問題ありません。そのため最小限の設定としています。

まず、インストールができているか確認します。

以下のコマンドを実行後、同様な表示があればインストールできています。もし、異なる表示であればインストールがうまくいっていない可能性があります。前のページに戻り再度インストールを試みます。

```
# systemctl status httpd
○httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/     ・・・・・)
     Active:   inactive (dead)
       Docs:   man:httpd.service(8)
```

何も設定していなくても動作します。実際に動作させ、確認します。

```
# systemctl start httpd
# systemctl status httpd
○httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/     ・・・・・)
     Active:   active (running)
       Docs:   man:httpd.service(8)
 Main PID: 55890 (httpd)
```

CentOS には、ブラウザとして firefox が事前にインストールされています。これを起動し、アドレスバーに CentOS の IP アドレス(192.168.41.31)を入力し、確認します。

確認ができれば、CentOS 再起動後、Web サーバーも自動的に起動する設定にします。

```
# systemctl enable httpd          < 自動起動設定
# systemctl is-enable httpd       < 自動起動になっているか確認
enabled
```

CentOS は、インストール直後、ファイアウォール(firewalld)が動作しています。操作端末から接続することができません。今回の実習では、閉じられたネットワーク(LAN 内)で行うため、ファイアウォールを停止させることとします(運用上好ましくないため実行環境において調整します)。

```
# systemctl stop firewalld          < firewalld 停止
# systemctl disable firewalld       < CentOS 再起動時、自動起動させない
```

操作端末(Android タブレット)のブラウザを使用し、接続を行います。利用するブラウザは、Google Chrome(以下、Chrome)である必要があります。

以下は、操作端末の Chrome を利用し、接続した画面になります。

さらに、CentOS では、SELinux と呼ばれるシステム内のファイル、ディレクトリなどを改変から守る仕組みが自動起動しています。これも同様の理由で、停止させます。

```
# vi /var/selinux/config
```

```
SELINUX=enforcing
        ↓
SELINUX=disabled          < disabled に変更
```

上書き保存後、CentOS を再起動させます。再起動をしないと設定が反映されません。

この設定により起動時に SELinux が自動起動しないようになります。

Web サーバー(Apache)の設定を変更していきます。

設定ファイルは、/etc/httpd/conf/httpd.conf　です。

設定を間違えた際に元に戻すことができるように、バックアップコピーを作成しておきます。

```
# cp /etc/httpd/conf/httpd.conf /etc/httpd/conf/httpd.conf.org
```

本来の運用であれば、たくさんの書き換えが必要ですが、実習に必要な個所だけとします。

実習用に設定変更を行います(今回は、vi エディタを利用していますが、使い慣れたエディタがあれば使って構いません)。

```
# vi /etc/httpd/conf/httpd.conf
```

98 行目を確認します。

```
#ServerName www.example.com:80
ServerName 192.168.41.31:80
```

99 行目が空行になっているので、そこに ServerName の記載を行います。

DNS サーバーを用意していないので、IP アドレスをそのまま記述します。

166 行目を確認します。

```
<IfModule dir_module>
        DirectoryIndex index.html index.php
</IfModule>
```

index.php を追記します。

上記2か所の変更ができれば、上書き保存をします。

記述に間違いがないか確認します。

```
# httpd -t
Syntax OK              < これ以外のメッセージであれば、修正を行います
```

4.8　Web サーバー、SSL サーバーの連携

Web Speech API を利用するためには、Web サーバー、SSL サーバーが連携し、https を利用した接続を行う必要があります。連携については、Apache に専用モジュールがインストールされていれば、特に変更なく動作します（本来の運用では鍵、証明書作成などきちんと調整したほうがよいです）。

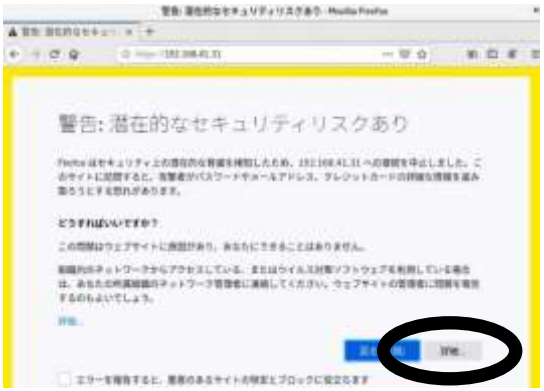
Apache と SSL が連携するモジュールがインストールされているか確認します。

```
# httpd -M | grep ssl
 ssl_module (shared)
```

表示がなければインストールされていません。前のページに戻りインストールします。

ブラウザ（firefox）のアドレスバーに、https://192.168.41.31/ を記述します。

以下のような警告画面が現れます。これは安全な接続を保証できないためです。今回は、LAN 内の Web サーバーに接続していますので、安全です。右下にある詳細を選択します。



さらに、確認を求められます。危険性を承知で続行 を選択します。

表示されます。アドレスバーには、https://192.168.41.31/ の表記になっています。

4.9 PHP の動作確認

Apache と PHP の連携動作を確認します。

PHP がインストールされているかを確認します。

```
# php -v
PHP 7.2.11 (cli) (built: Oct  9 2018 15:09:36) ( NTS )
Copyright ( c) 1977-2018 The PHP Group
Zend Engine v3.2.0, Copyright ( c) 1998-2018 Zend Technologies
```

上記のように PHP のバージョン情報が出てこない場合、インストールされていません。前のページに戻り、インストールします。

Apache では、PHP との連携が設定変更なく利用できる状態です。PHP のファイルを作成し、動作確認を行います。

```
# vi /var/www/html/info.php
```

内容としては1行のみ記述し、保存します。

```
<?php phpinfo(); ?>
```

ブラウザのアドレスバーに、 https://192.168.41.31/info.php を入力します。

もし表示できないようであれば、Apache を再起動させてください。

```
# systemctl restart httpd
```

5　ホームページ作成

Web Speech API は、Web サーバー経由で動作させます。そこで、操作用の簡単なホームページを作成する必要があります。API を動作させるために、最低限 HTML、JavaScript の記述が必要になります。この2つの記述方法を理解します。

また、操作端末（タブレットなど）から Web ページ経由で ESP32 に動作指示を出す際に、指示内容を処理することになります。そこでは、PHP スクリプトを利用しています。そのため PHP についても理解します。

このことをシステム全体でまとめると以下のようになります。

- HTML は、表示（見た目）
- JavaScript は、Web Speech API 呼び出し
- PHP スクリプトは、ESP32 に渡す制御情報を処理

5.1　HTML（HyperText Markup Language）

1989年にHTMLが開発され、バージョンアップが繰り返されています。2014年からHTML5が運用され、各種ブラウザで表示が可能になりました。現在では、HTML5 の記述方法に基づいて記述することになります。

本来であれば、HTML、CSS（Cascading Style Sheets）を利用し、デザイン、レイアウトなどの調整を行いますが、今回は ESP32 との連携に必要な記述のみを確認します。

ホームページ作成

HTML は、テキストエディタで記述します。HTML の記述がしやすいエディタなどありますので、そちらを利用しても構いません。

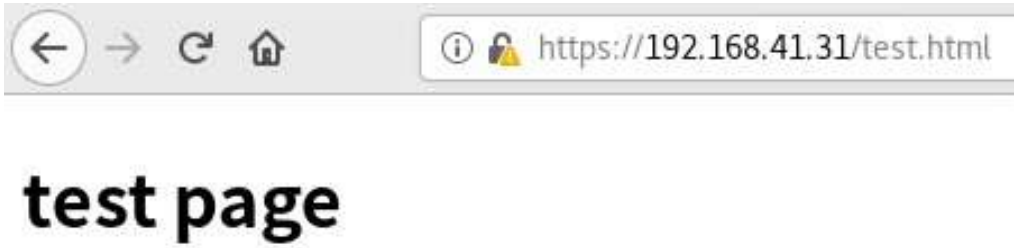
今回は、あまりたくさんの記述は行わないため CentOS 上の vi エディタ を利用します。Apache の表示用ディレクトリは、/var/www/html/ のため、この中に HTML ファイルを作成します。

```
# vi /var/www/html/test.html
```

内容として、以下を記述します。記述できれば保存します。

```
<!DOCTYPE html>                 < HTML5 で記述されていること表します
<html lang="ja">                < 日本語環境である
<head>
  <meta charset="UTF-8">        < 文字コードは UTF-8 を利用している宣言
</head>
<body>
 <h1>test page</h1>             < 閲覧者に表示する内容
</body>
</html>
```

ブラウザのアドレスバーに、https://192.168.41.31/test.html と入力すると、以下のように表示されます。



test page

ホームページ作成

次に、Web ページ上にボタンを作成します。

ボタンを表示するための記述は、以下のようになります。

```
<button id=”識別情報”>ボタン表面の文字列</button>
```

複数のボタンを表示させると識別できなくなるため id=”識別情報” により、各ボタンを識別します。識別情報は、重複しないように記入します。

先ほど作成した HTML ファイルに追記します。

```
# vi /var/www/html/test.html
```

```
<!DOCTYPE html>
<html lang=”ja”>
<head>
  <meta charset=”UTF-8”>
</head>
<body>
 <h1>test page</h1>
 <button id=”btn1”>button1</button>          < ボタンを作成
 <button id=”btn2”>button2</button>          < ボタンを作成
</body>
</html>
```

ブラウザを更新すると、ボタンが2つ表示されます。

次に、Web ページ上にテキストボックスを作成します。

テキストボックスを表示するための記述は、以下のようになります。

```
<input type="text" id="識別情報">
```

複数のテキストボックスを表示させると識別できなくなるため id="識別情報" を利用します。識別情報は、すべて重複しないように記入します。

value="abc"のような記述を追加すると、事前にテキストボックス内に入力情報 abc を入れておくことができます。

先ほど作成した HTML ファイルに追記します。

```
# vi /var/www/html/test.html
```

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
</head>
<body>
 <h1>test page</h1>
 <button id="btn1">button1</button>          < ボタンを作成
 <button id="btn2">button2</button>          < ボタンを作成
 <br><br>                                    < 改行します
 <input type="text" id="strdata">            < テキストボックスを作成

</body>
</html>
```

ブラウザを更新すると、テキストボックスが表示されます。

5.2 JavaScript

Web Speech API を動作させるためには、JavaScript が必要になるためです。

JavaScript は、HTML だけではできない動きのあるホームページを作成することができます。1996 年から開発が始まり、現在では、主要な言語となっておりホームページ作成以外でも利用されています。

こちらも ESP32 との連携に必要な内容のみに絞って、記述、動作確認を行います。

JavaScript を記述するためには、HTML ファイル内に記述、別ファイルに記述する2つの方法があります。まず、HTML ファイル内に記述する方法から動作させます。

HTML ファイル内に記述するには、以下のようになります。

```
<script>
    JavaScript のコードを記述する
</script>
```

先ほど作成した HTML ファイルに追記します。

```
# vi /var/www/html/test.html
```

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
</head>
<body>
 <h1>test page</h1>
 <button id="btn1">button1</button>
 <button id="btn2">button2</button>
 <script>
    document.write("JavaScript");                < JavaScript の記述
</script>
</body>
</html>
```

もし表示されなければ、記述が間違っています。再度、test.html を確認します。

5.2.1　イベントハンドラの利用

JavaScript では、閲覧者がマウス、キーボード操作などを行った際、Web ページ上に変化を付けることができます。このマウス操作などを イベント と言います。このイベントと共に JavaScript のプログラムを動作させることを イベントハンドラ と言います。

この機能を利用して、ボタンが押されると表示に変化を付けます。

先ほど作成した HTML ファイルに追記します。

```
# vi /var/www/html/test.html
```

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
 <script>                              < JavaScript の記述
   function doChange1() {              < doChange1 関数の作成
      document.getElementById("h1").innerHTML = "Click";
   }                                   < id が h1 の表示を Click に変更する
 </script>
</head>
<body>
 <h1 id="h1">test page</h1>                < id 情報を追加
 <button id="btn1" onclick="doChange1()">button1</button>
 <button id="btn2">button2</button>
</body>
</html>
```

ボタン 1 をクリックすると、表示が Click に変化します。

onclick="doChange1()" により、クリックされたら doChange1()関数 を呼び出すことになります。

document.getElementById("h1").innerHTML = "Click" により、id="h1" の記述のある情報を Click に変更するという意味になります。

同様のプログラムを別ファイルにして、実行してみます。

別ファイルに記述する際には、HTML ファイル内に以下のように記述します。

```
<script src=JavaScript ファイルのパス></script>
```

これを踏まえて、先ほど作成した HTML ファイルを変更します（JavaScript の記述を削除）。

```
# vi /var/www/html/test.html
```

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <script src="testscript01.js"></script>        < JavaScript を別ファイルに
</head>
<body>
  <h1 id="h1">test page</h1>
  <button id="btn1" onclick="doChange1()">button1</button>
  <button id="btn2">button2</button>
</body>
</html>
```

JavaScript ファイルを作成します。

```
# vi /var/www/html/testscript01.js
```

```
function doChange1() {
  document.getElementById("h1").innerHTML = "Click";
}
```

動作は同様です。JavaScript の記述が多くなると HTML の記述と混ざり可読性も悪くなるため、別ファイルに記述することが多いです。

5.2.2　変数の利用

JavaScript では、変数を利用することができます。しかし、C 言語などのプログラム言語とは異なり、型の考え方がないため一度利用を宣言するとどのような値でも代入することができます。

変数の宣言と代入方法は、以下のようになります。

```
<script>
    let data1 = 123;        < 利用宣言は、let。宣言と共に整数を代入している
    data1 = ”abc”;          < 後から異なる(文字列)型を代入することが可能
</script>
```

あまり良い例ではありませんが、整数型で代入していた変数に、後から文字列型を代入することも可能です。これは、宣言時に型を指定する必要がないからです。

先ほど作成した JavaScript ファイルを編集します。

```
# vi /var/www/html/testscript01.js
```

```
let dataDisplay = ”Click”;              < 変数宣言
function doChange1() {
  document.getElementById(”h1”).innerHTML = dataDisplay;    < 変数利用
}
```

これも動作結果は変わりません。

5.2.3　制御構文

JavaScript では制御構文には、一般的に以下のものが利用されています。

- if 構文
- switch 構文
- for 構文
- while 構文

今回の実習で必要な構文は、if 構文のみです。そのため if 構文のみ確認を行っていきます。
基本構文は、以下のようになります。

```
<script>
    if（ 条件判断 ）{
        true 時の処理内容
    } else {                          < false の処理内容がなければ、省略可
        false 時の処理内容
    }
</script>
```

複数条件により処理を行う場合は、以下のようになります。

```
<script>
    if（ 条件判断 1 ）{
        条件判断 1 が、true 時の処理内容
    } else if（ 条件判断 2 ）{
        条件判断 1 が false、条件判断 2 が true 時の処理内容
    } else {
        条件判断 1、2 共に false 時の処理内容
    }
</script>
```

条件判断に用いる比較演算子、論理演算子は、C 言語などと違いはありません。

＝＝	等価比較	&&	And 演算子
！＝	不等価比較	||	Or 演算子
＜	より小さい	！	Not 演算子
＜＝	以下		
＞	より大きい		
＞＝	以上		

以上のことを踏まえ、プログラムを修正します。先ほど作成した HTML ファイルを変更します。

```
# vi /var/www/html/test.html
```

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
 <script src="testscript01.js"></script>
</head>
<body>
 <h1 id="h1">test page</h1>
 <button id="btn1" onclick="doChange1()">button1</button>
 <button id="btn2" onclick="doChange2()">button2</button>
</body>
</html>
```

先ほど作成した JavaScript ファイルに追記します。

```
# vi /var/www/html/testscript01.js
```

```
let dataDisplay = "Click";
function doChange1() {
  document.getElementById("h1").innerHTML = dataDisplay;
}

let dataNum = 123;                    < 変数 dataNum に、123 を代入
function doChange2() {
  if ( dataNum > 100 ) {              < 変数 dataNum が 100 よりも大きいか？
    document.getElementById("h1").innerHTML = "100+";
  } else {
    document.getElementById("h1").innerHTML = "-100";
  }
}
```

ボタン2をクリックすると、100+　の表示が現れます。比較する変数（dataNum）の値を変更すると結果を変えることができます。

5.2.4　ページ遷移と GET メソッド

HTML ファイルにより表示されているページを、JavaScript などを利用し、別のページに移動させます。この時に、別のページに移動する際に付加情報を渡します。

この付加情報を利用し、ESP32 に動作指示を出します。

付加情報を追加する方法は、主に2つ方法があり、どちらかを利用します。

- GET メソッド
- POST メソッド

GET メソッドを利用したものは、ブラウザのアドレスバーに URL の後ろに付加情報を追加したものです。

GET メソッドの例 ： https://192.168.41.31/test.html?data1=5&data2=abc

GET メソッド情報は、？以降に　名前＝値　を付加したものです。このデータの渡し方になると利用者が勝手に情報を変更できたりするため注意が必要です。パスワードなどの重要な情報を送るためには使いません。しかし、簡単に作成できます。

POST メソッド情報は、アドレスバーに付加情報が表示されません。そのため情報を隠すことができ、また GET メソッドより多くの情報を渡すことができます。しかし、ヘッダー情報を確認することができれば、中身は平文で送られているため、やはりパスワードなどの情報をそのまま送ることには向きません。

今回の実習では、GET メソッドを利用した方法でページ遷移を行います。あくまで、LAN 内での実行を想定しており、作成も容易にできるためです。

JavaScript を利用して、ページ遷移を行う場合は、以下のようになります。

```
<script>
    location.href = URL  または、相対パス
</script>
```

ページ遷移を行う場合に、GET メソッド付きで行う場合は、以下のようになります。

```
<script>
    location.href = URL?変数名=値        < URL？ の後に GET 情報を追加します
</script>
```

変数名＝値で設定しますが、＝の前後にスペースを空けないように注意してください。

遷移先の HTML ファイルを作成します。

```
# vi /var/www/html/data.html
```

```
<!DOCTYPE html>
<html lang=”ja”>
<head>
  <meta charset=”UTF-8”>
</head>
<body>
 <h1 id=”h1”>data page</h1>
</body>
</html>
```

先ほど作成した JavaScript ファイルを変更します。

```
# vi /var/www/html/testscript01.js
```

```
let dataDisplay = ”Click”;
function doChange1() {              < ボタン 1 がクリックされるとページ遷移
  location.href = ”data.html?data1=789”;       < GET メソッドを付加
}

let dataNum = 123;
function doChange2() {
  if ( dataNum > 100 ) {
    document.getElementById(”h1”).innerHTML = ”100+”;
  } else {
    document.getElementById(”h1”).innerHTML = ”-100”;
  }
}
```

test.html を表示させ、ボタン1を選択すると、GET メソッド付きでページ遷移を行います。

同一ページに GET メソッド付きで遷移もできます。JavaScript ファイルを変更します。

```
# vi /var/www/html/testscript01.js
```

```
let dataDisplay = ”Click”;
function doChange1() {
  location.href = ”data.html?data1=789”;
}

let dataNum = 123;
function doChange2() {          < ボタン 2 が押されたら GET メソッド
  location.href = ”?data1=789”;          < URL の指定をしないと同一ページ
}
```

ボタン2を選択すると、test.html(同一ページ)に GET メソッド付きで送信します。

5.3 PHP スクリプト

PHP は、Web アプリケーションを開発するために利用されています。アプリケーションを開発する際、記述方法が他の言語に比べると書きやすく、開発が行いやすい特徴を持っています。Web アプリケーション開発をするプログラムのため、様々な機能が用意されています。

詳しい内容は省略し、今回の実習に必要な内容のみに絞って説明を行います。

PHP を利用することにより、JavaScript から送られた GET メソッドを取得し、取得した情報をテキストファイルに書き込みます。テキストファイルに書き込まれた情報により ESP32 を動作させます。

PHP の記述は HTML ファイル内に埋め込む形で実行させます。そのためスクリプトという言葉が使われています。PHP の記述があるファイルの拡張子は、.php にする決まりになっています。

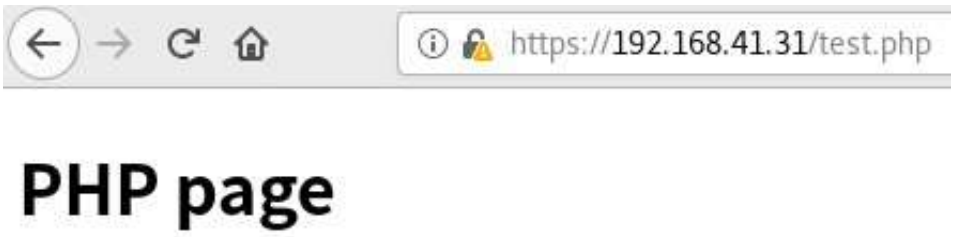
簡単な PHP プログラムを記述し、実行します。

PHP ファイルを新規作成します(拡張子を php に)。

```
# vi /var/www/html/test.php
```

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
</head>
<body>
 <h1 id="h1"><?php print("PHP page"); ?></h1>
</body>
</html>
```

ブラウザで表示すると PHP で記述した内容が表示されます。



PHP のプログラムを記述する際は、<?php PHP プログラム ?> 内に記述することになっています。

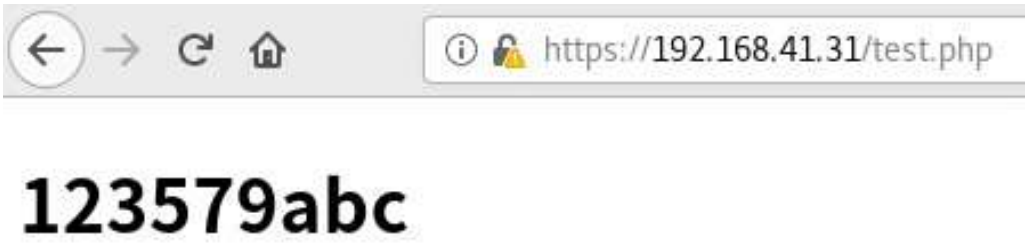
5.3.1　変数の利用

PHP においても変数が利用できます。PHP も JavaScript と同じく変数の型の概念がありません。変数にどのような値でも入れることができます。そのため、後から型の違ったものを代入することもできます。

変数を利用する場合、先頭に　$　を付けなければならない決まりです。
例としては、以下になります。先ほど作成したファイルを変更します。

```
# vi /var/www/html/test.php
```

```
<!DOCTYPE html>
<html lang=”ja”>
<head>
  <meta charset=”UTF-8”>
</head>
<body>
 <h1 id=”h1”>
<?php
  $abc = 123;        < 変数 abc を利用
  print($abc);
  $abc = $abc + 456;      < 計算もできます
  print($abc);
  $abc = ”abc”;     < 全<異なる型を代入することも可能
  print($abc);
?>
  </h1>
</body>
</html>
```



5.3.2　テキストファイルの操作

JavaScript では、テキストファイルなどのプログラムが記述されている以外の外部ファイルを扱うことができません。そこで、PHP を利用して外部ファイルに情報の読み出し、書き込みを行います。

動作確認を行うためにテキストファイルを作成し、事前に情報を書き込んでおきます。

```
# vi /var/www/html/dataphp.txt
```

```
External file
Content php
```

書き込み対象ファイルの所有者が apache ユーザーでなければ、書き込みができません。所有者を変更します。

```
# chown apache:apache /var/www/html/dataphp.txt
```

PHP のプログラムを変更します。

```
# vi /var/www/html/test.php
```

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
</head>
<body>
 <pre>       < 改行をそのまま表示するために、h1 から変更します
<?php
  $filepath = "dataphp.txt";
  $data = file_get_contents($filepath);          < ファイルの中身を読み出し
  print($data);

  file_put_contents($filepath, "abcde¥n");       < ファイルに書き込み
?>
  </pre>
</body>
</html>
```


```
External file
Content php
```

```
[root@localhost ~]# cat /var/www/html/dataphp.txt
abcde
```

PHP でのファイル操作には、
ファイルの中身を読み出す場合には、

file_get_contents(ファイルパス);

ファイルに書き込む場合には、

file_put_contents(ファイルパス, 書き込みたい文字列);

ファイルを読み出すことができない事はあまりありませんが、書き込みができない場合は、対象ファイルのアクセス権が適切な状態でない場合があります。再度、確認してみてください。

5.3.3 GET メソッドの取得

PHP は、GET メソッドで送られたデータを受け取り、プログラム内の処理に利用することができます。

GET メソッドを取得する場合、PHP の定義済み変数 $_GET[key 名] を利用します。

5.2.4 の JavaScript を利用し GET メソッド送信を行い、送信された GET メソッドを PHP で取得、処理を行います。

以上のことを踏まえ、test.html をコピーし、PHP ファイルを作成します。

```
# cp /var/www/html/test.html /var/www/html/testget.php
# vi /var/www/html/testget.php
```

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <script src="testscript01.js"></script>
</head>
<body>
 <h1 id="h1">
  <?php

    if (isset($_GET['data1'])) {          < GET メソッド data1 が存在すれば
      print($_GET['data1']);              < data1 の 値を表示
    } else {
      print("test page");
    }

  ?>
 </h1>
 <button id="btn1" onclick="doChange1()">button1</button>
 <button id="btn2" onclick="doChange2()">button2</button>
</body>
</html>
```

PHP の制御構文を利用していますが、JavaScript と同様の記述方法になります。

同一ページに GET メソッド付きで遷移します。JavaScript ファイルを確認します。

```
# vi /var/www/html/testscript01.js
```

```
let dataDisplay = ”Click”;
function doChange1() {
  location.href = ”data.html?data1=789”;
}

let dataNum = 123;
function doChange2() {              < ボタン 2 が押されたら GET メソッド
  location.href = ”?data1=Done_Click”;
}
```

ボタン2をクリックすると同一ページに GET メソッドを送信します。

受信した GET メソッドの値に基づいて、表示が変わります。

6　Web Speech API

音声認識システムの構成するために、Web Speech API を動作させます。基本的に、音声認識、音声合成の2つの仕組みから構成されています。Web Speech API の呼び出しには、JavaScript が必要になります。

以下の2つの機能が、Web ブラウザに搭載されていなければ動作させることができません。

- 音声認識(音声を文字にする)は、SpeechRecognition
- 音声合成(文字を音声にする)は、SpeechSynthesis

以下の図3が、SpeechRecognition 対応状況になります。

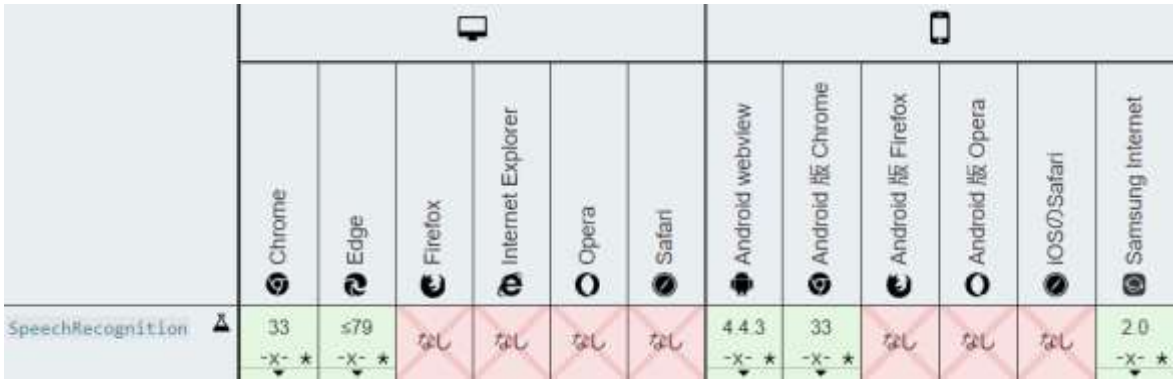


図 3：SpeechRecognition 対応状況　※5

以下の図 4 が、SpeechSynthesis 対応状況になります。

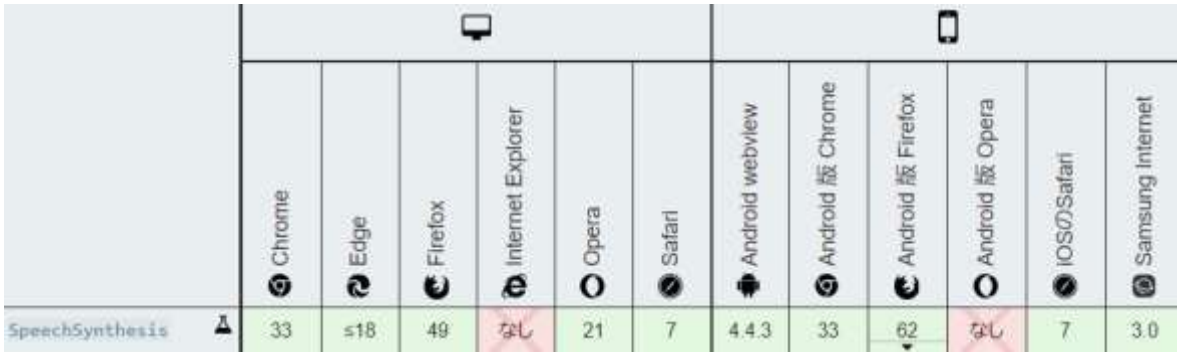


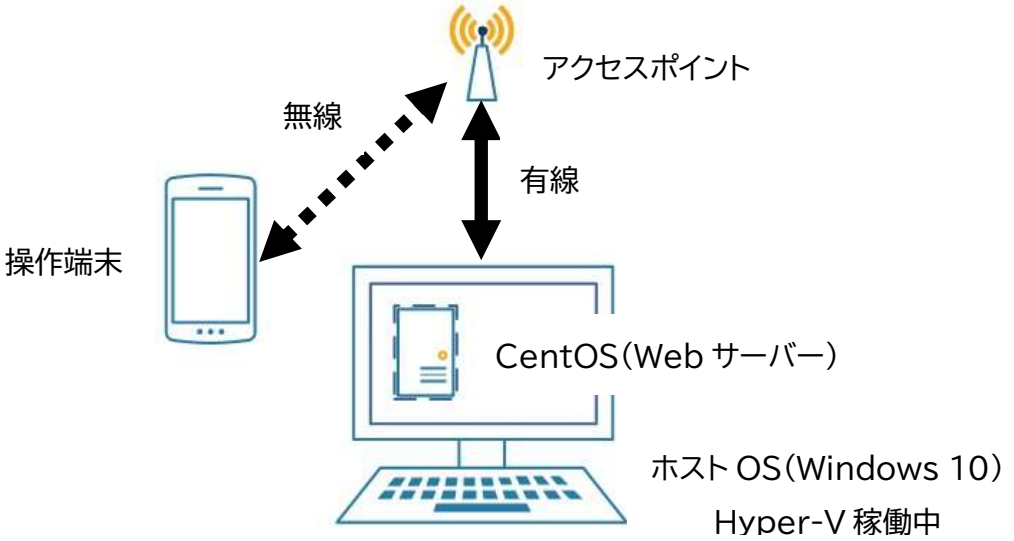
図 4：SpeechSynthesis 対応状況　※5

筆者の方で、各種 Web ブラウザにおいて動作確認を行うと、Google Chrome が最も動作がよいと判断しました。そのため今回の実習では、Web Speech API の動作は、Android タブレットを利用し、Google Chrome にて動作確認を行うことにしました。

その他、詳しい情報は、以下の URL で確認してください。

URL：https://developer.mozilla.org/ja/docs/Web/API/Web_Speech_API

この後の実習の動作環境としては、以下の構成で実行しています。



ネットワーク体系は、以下のように設定しています。サブネットマスクは、24bit です。

	IP アドレス
アクセスポイント	192.168.41.1
ホスト OS（Windows 10）	192.168.41.30
CentOS（Web サーバー）	192.168.41.31
操作端末	192.168.41.100

アクセスポイント設定は、以下のようにしています。

SSID	speechapidemo
パスワード（WPA2）	polytechtest

注意点として、操作端末のみインターネットに接続する必要があります。動作確認を行うため操作端末の音量を上げておきます。

通信環境については、実際の実習環境において適宜変更してください。

6.1　音声合成 SpeechSynthesis の利用１ ※6

JavaScript を利用し、SpeechSynthesis を呼び出し、操作端末に発声させます。

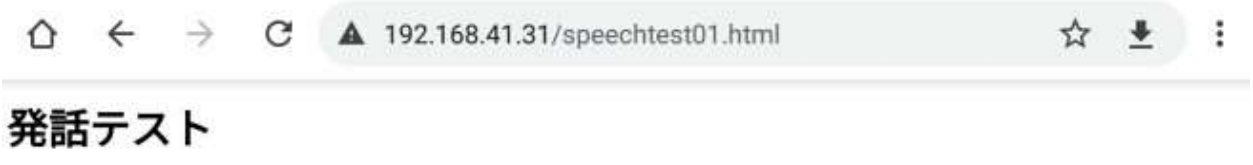
Web ページが表示されると、「こんにちは」 と操作端末のスピーカーから発声します。
HTML ファイルを作成します。

```
# vi /var/www/html/speechtest01.html
```

```
<!DOCTYPE html>
<html lang="ja">
<head> <meta charset="UTF-8"> </head>
<script>

// 音声合成 API の使用
 let speech = new SpeechSynthesisUtterance();

// 言語を日本語に設定
 speech.lang = "ja-JP";

// 発声させるキーワード(文字列)を指定
 speech.text = "こんにちは";

// speechSynthesis の呼び出し(発声させる)
 speechSynthesis.speak( speech );

</script>
<body>
  <h1 id="h1">発話テスト</h1>
</body>
</html>
```

下記のように、発声したい内容を指定することにより、指定の文字列を発声します。
speechSynthesis.speak(new SpeechSynthesisUtterance("発声する内容"));

以下のように操作端末で https://192.168.41.31/speechtest01.html を表示すると同時に、「こんにちは」 と発声してくれます。



6.2　音声合成 SpeechSynthesis の利用 2

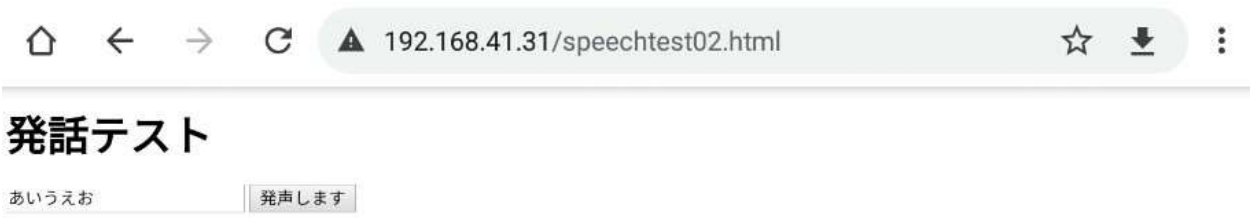
テキストボックスを表示させ、その中に文字を入力し、入力された文字を読み上げます。

HTML ファイルを作成します。

```
# vi /var/www/html/speechtest02.html
```

```
<!DOCTYPE html>
<html lang="ja">
<head> <meta charset="UTF-8"> </head>
<script>

// 音声合成 API の使用
 let speech = new SpeechSynthesisUtterance();

// 言語を日本語に設定
 speech.lang = "ja-JP";

function btnspeech() {              // 発声しますボタンをクリックすると

// テキストボックスの値を取得
    speech.text = document.getElementById("strdata").value;
// テキストボックス内の値を発声させる
    speechSynthesis.speak(speech);
}
</script>
<body>
 <h1 id="h1">発話テスト</h1>
 <input type="text" id="strdata" max="20">            < テキストボックス表示
 <button onclick="btnspeech()">発声します</button>
</body>
</html>
```

以下のように、テキストボックスが表示されます。そこに、日本語を入力します。その後、発声しますボタンを選択すると、テキストボックス内の文字を読み上げてくれます。



6.3　音声認識 SpeechRecognition の利用1 ※7

音声合成 SpeechSynthesis と同様に、JavaScript を利用し、SpeechRecognition を呼び出し、音声を認識させます。

音声認識に関しては、Google Cloud Speech API を利用しているらしく、利用する端末をインターネットに接続する必要があります。しかし、いくら利用しても料金はかかりません。

HTML ファイルが表示されると音声認識を開始し、認識した情報をページ内に表示させます。

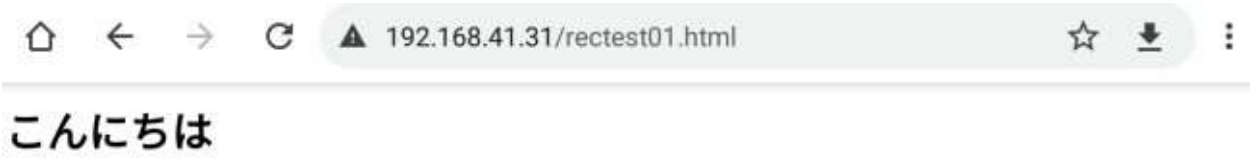
HTML ファイルを作成します。

```
# vi /var/www/html/rectest01.html
```

```
<!DOCTYPE html>
<html lang="ja">
<head> <meta charset="UTF-8"> </head>
<script>

// 音声認識 API の使用
 let speechrec = new webkitSpeechRecognition();
// 言語を日本語に設定
 speechrec.lang = "ja-JP";

// 認識した音声情報を表示させる
 speechrec.onresult = function(event) {

   // 音声認識した情報を変数 results に代入
     let results = event.results[0][0]. transcript;
     document.getElementById('h1').innerHTML = results;
}

// 音声認識開始
 speechrec.start();

</script>
<body>
  <h1 id="h1">音声認識テスト</h1>          < この部分の表示が認識した情報に変化
</body>
</html>
```

はじめてページにアクセスすると以下のように、マイクの使用許可を求められます。許可するようにします。

その後、操作端末へしゃべりかけます。しゃべりかけた内容を表示してくれます。



6.4　音声認識 SpeechRecognition の利用 2 ※8

ボタン1を押すと認識が始まり、ボタン2を押すと認識を停止します。

HTML ファイルが表示されると音声認識を開始し、認識した情報をページ内に表示させます。
HTML ファイルを作成します。

```
# vi /var/www/html/rectest02.html
```

```
<!DOCTYPE html>
<html lang=”ja”>
<head> <meta charset=”UTF-8”> </head>
<script>

// 音声認識 API の使用
 let speechrec = new webkitSpeechRecognition();
// 言語を日本語に設定
 speechrec.lang = ”ja-JP”;

// 認識した音声情報を表示させる
 speechrec.onresult = function(event) {

   // 音声認識した情報を変数 results に代入
     let results = event.results[0][0]. transcript;
     document.getElementById(’h1’).innerHTML = results;
}

// 音声認識開始
function rec_start() {
    speechrec.start();  }

// 音声認識停止
function rec_stop() {
    speechrec.stop();  }
</script>
<body>
 <h1 id=”h1”>音声認識テスト</h1>          < この部分の表示が認識した情報に変化
 <button onclick=”rec_start()”>開始</button>
 <button onclick=”rec_stop()”>停止</button>
</body>
</html>
```

開始、停止ボタンが表示されます。
開始ボタンを押すと認識を開始し、停止ボタンで停止します。その後、認識した情報を表示します。

こんにちは

開始 停止

6.5　音声認識、合成を組み合わせ、オウム返し

基本は、先ほど作成した音声認識を文字で表示するだけでなく、音声で返します。ボタン1を押すと認識が始まり、ボタン2を押すと停止します。

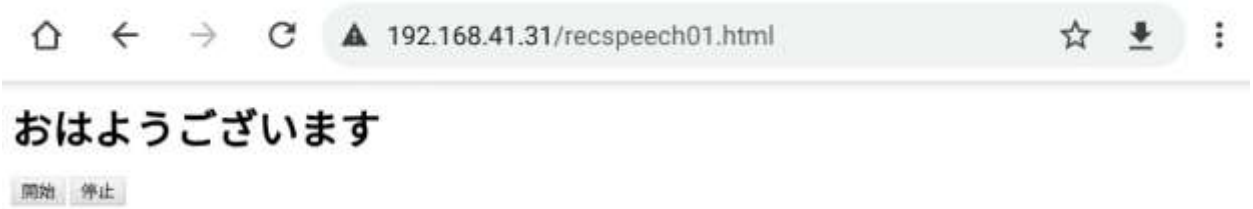
HTML ファイルを作成します。

```
# vi /var/www/html/recspeech01.html
```

```
<!DOCTYPE html>
<html lang="ja">
<head>  <meta charset="UTF-8">  </head>
<script>
// 音声合成 API の使用
 let speech = new SpeechSynthesisUtterance();
 speech.lang = "ja-JP";
// 音声認識 API の使用
 let speechrec = new webkitSpeechRecognition();
 speechrec.lang = "ja-JP";

// 認識した音声情報を処理
 speechrec.onresult = function(event) {
    // 文字表示
    let results = event.results[0][0]. transcript;
    document.getElementById('h1').innerHTML = results;
    // 発話
    speech.text = results;
    speechSynthesis.speak(speech);
}
// 音声認識開始
function rec_start() {
     speechrec.start();   }
// 音声認識停止
function rec_stop() {
     speechrec.stop();   }
</script>
<body>
 <h1 id="h1">オウム返しテスト</h1>
 <button onclick="rec_start()">開始</button>
 <button onclick="rec_stop()">停止</button>
</body>
</html>
```

以下のように表示され、開始ボタンを選択すると認識を開始します。

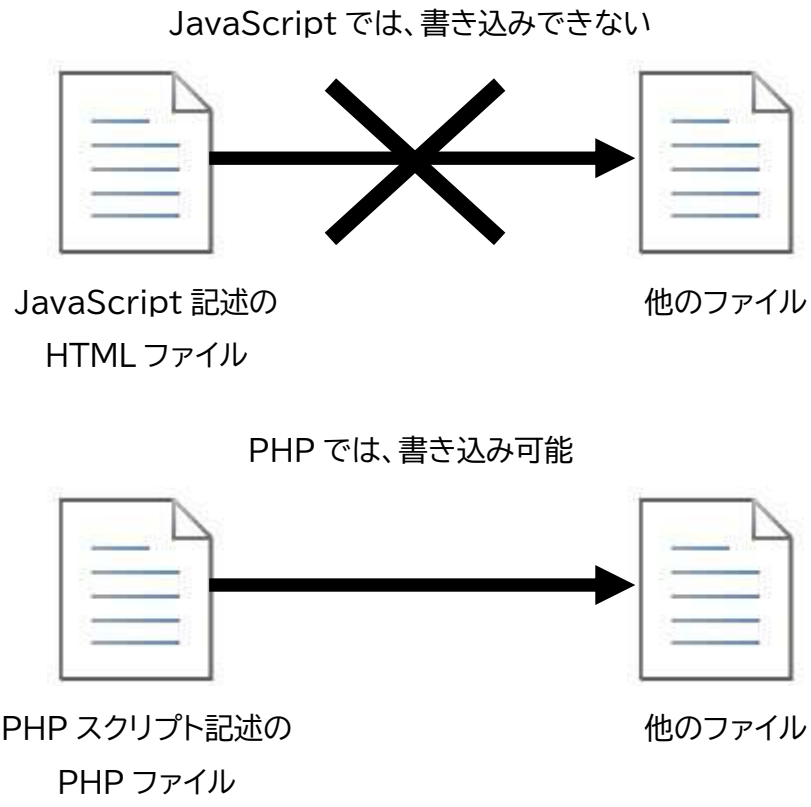
認識した情報を表示した後、発声します。



6.6　音声認識した情報をテキストファイルに書き込み

ESP32 制御のために音声認識した情報をテキストファイルに書き込むことが必要になります。

これを実行するためには、JavaScrip だけではできません。セキュリティ上の観点から JavaScript では、実行中のファイル以外を操作できないようになっています。そこで、PHP を利用し、受信した情報をテキストファイルに書き込みます。



GET メソッドを利用し、GET メソッドで送信された内容を書き込みます。

テキストファイルを作成し、その中に書き込めるようにします。中身は、空で構いません。

```
# touch /var/www/html/phptest.txt
```

以前も確認しましたが、書き込み対象ファイルの所有者が apache ユーザーでなければ、書き込みができません。所有者を変更します。

```
# chown apache:apache /var/www/html/phptest.txt
```

JavaScript を用いて、音声認識 API を実行、認識した情報を GET 送信、PHP で GET 受信後にテキストファイルに書き込むプログラムを作成します。

```
# vi /var/www/html/rectext.php
```

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <script>
// 音声認識 API の使用
 let speechrec = new webkitSpeechRecognition();
 speechrec.lang = "ja-JP";

// 認識した音声情報を処理
 speechrec.onresult = function(event) {
   let results = event.results[0][0]. transcript;

   if (results == "") {
       location.reload();            < 情報がなければ、再読み込み
   } else {
       location.href = "?data1=" + results;      < 情報があれば、GET メソッド送信
  }
}

// 音声認識開始
function get_send() {
    speechrec.start();
}
  </script>
</head>
```

次のページに続きます。

```
<body>
<h1> GET メソッドによるテキストファイル書き込みテスト </h1>
<button onclick="get_send()">GET 送信</button>

<?php
  $filepath = "phptest.txt";
  $espdata = file_get_contents($filepath);      < ファイルの中身を読み出し

  if (isset($_GET['data1'])) {                  < GET メソッド data1 が存在すれば
      $espdata = $_GET['data1'];                < data1 の値を espdata に格納
  } else {
      $espdata = "";                            < なければ、中身を消す
  }

  file_put_contents($filepath, $espdata);       < ファイルに書き込み
?>

</body>
</html>
```

操作端末から Web サーバーに接続し、GET 送信　ボタンを選択します。

選択後、何かしゃべりかけます。音声認識を行い、GET 送信を実行しています。

GET 送信された情報は、テキストファイル（phptest.txt）内に書き込まれます。

```
[root@localhost ~]# cat /var/www/html/phptest.txt
おはよう[root@localhost ~]#
```

6.7　その他機能について

ここまで挙げた内容以外にも、Web Speech API には様々な音声関連機能が用意されています。これらを利用することも可能です。

音声合成であれば、以下の機能などがあります。

- 読み上げ中に一時停止、再開
- 読み上げ速度の変更
- 音の高さ変更
- 読み上げ言語変更　　など

音声認識であれば、会議の文字起こしのように、会議中に長時間認識させ続けることも可能です。

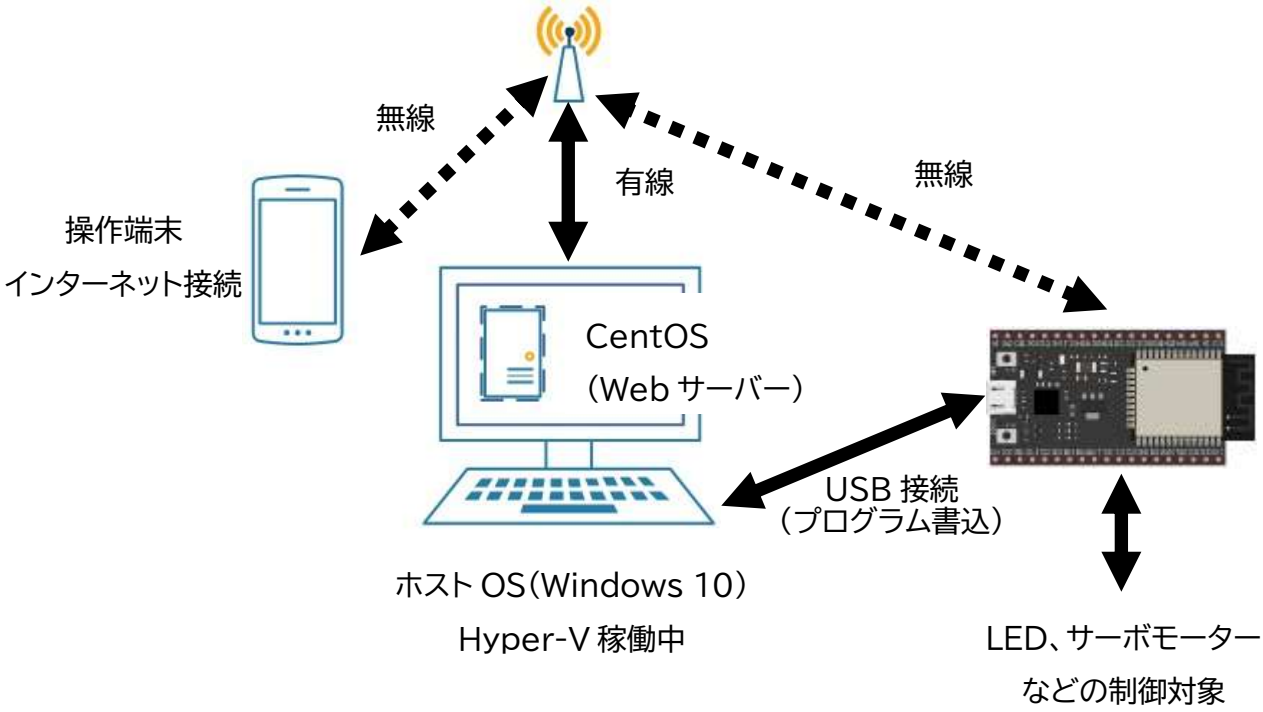
その他、様々な機能がありますが、今回の実習に必要な範囲ではないため割愛させていただきます。公式サイトの方にたくさんの情報があるのでぜひ確認してみてください。

URL : https://developer.mozilla.org/ja/docs/Web/API/Web_Speech_API

URL : https://wicg.github.io/speech-api/

7　ESP32 音声制御

いよいよ ESP32 を音声制御していきます。
実行環境を以下のようにしています。



ネットワーク体系は、以下のようにしています。サブネットマスクは、24bit です。

	IP アドレス
アクセスポイント	192.168.41.1
ホスト OS（Windows 10）	192.168.41.30
CentOS（Web サーバー）	192.168.41.31
ESP32	192.168.41.32
操作端末	192.168.41.100

アクセスポイント設定は、以下のようにしています。

SSID	speechapidemo
パスワード（WPA2）	polytechtest

注意点として、操作端末のみインターネットに接続する必要があります。

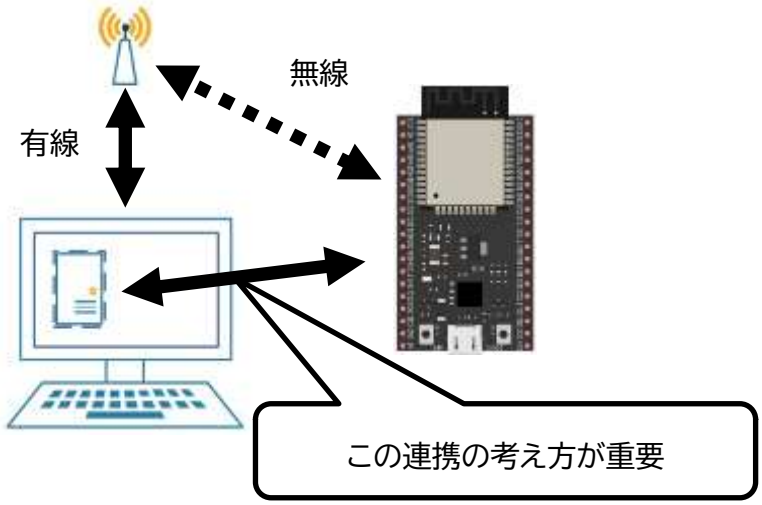
通信環境については、実際の実習環境において適宜変更してください。

7.1　プログラム動作概要

ハードウェア制御を行うにあたり、動作概要とプログラムを確認します。動作を以下のようにしています。

1. Web サーバーを起動させ、Web ページを表示可能にする
2. 利用者は、操作端末から Web サーバーに接続する
3. 利用者は、操作端末へキーワードを声で伝える
4. Web サーバーでキーワードを受信する
5. CentOS 内のテキストファイルに、キーワードを書き込む
6. ESP32 で Web サーバーに接続し、テキストファイルを読み出す

ポイントとなる内容は、6 の個所です。ESP32 と Web サーバーとの連携についてです。



Web サーバー、ESP32 ハードウェア制御情報を送受信するやり方としていくつか方法が検討できます。筆者が検討した内容が以下になります。

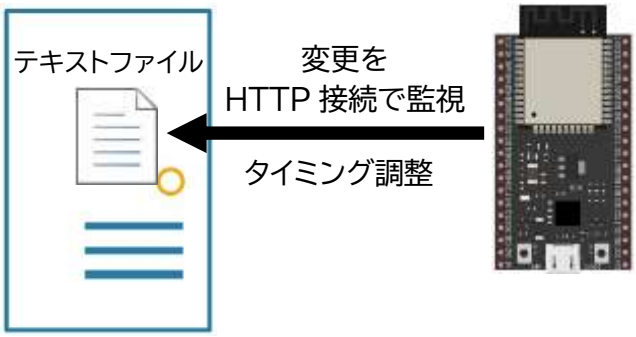
- MQTT（Message Queue Telemetry Transport）の利用
- クラウドシステム（IFTTT（if this then that）、AWS など）の利用
- HTTP を利用してテキストファイルを監視

MQTT を利用すると Web サーバーにおける変化を読み取り自動的に ESP32 に渡すことができます。しかし、MQTT サーバーの構築、ESP32 においてプログラムによる接続環境を構築するなどハードウェア制御動作までのハードルが少し上がります。

クラウドシステムを利用することになれば、契約等の問題もあります。

そこで、PHP を利用し GET メソッド受信によりテキストファイルを書き換えます。そのテキストファイルを ESP32 により常に監視します。ESP32 の HTTP 接続プログラムは比較的容易で、監視するタイミングを調整すると、速やかな制御も可能になるからです。今回の実習では、これを採用しました。

以下のような動作とします。

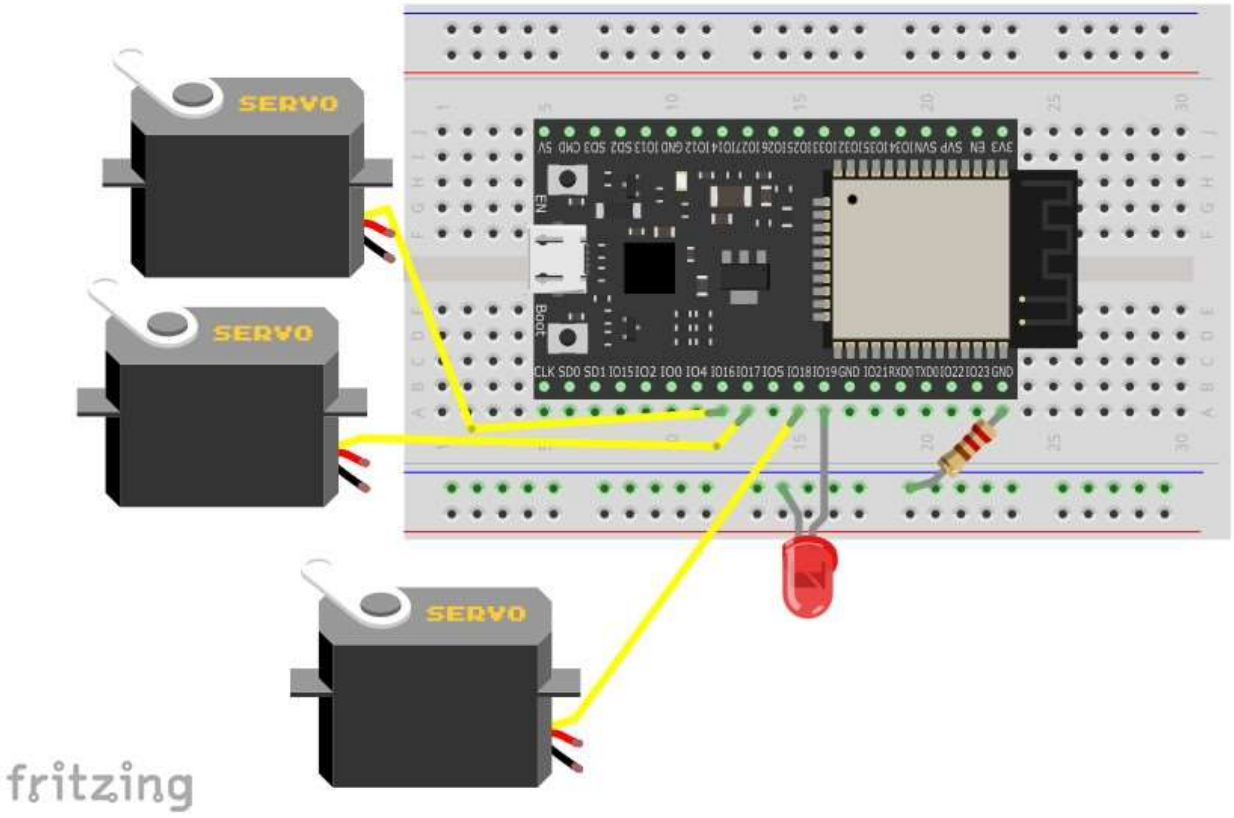


Web サーバー

もしセキュリティを考慮するようであれば、Web サーバー（Apache）の設定を変更し、対象のテキストファイルを ESP32 以外からは接続拒否するとよいでしょう。

7.2　ESP32 ハードウェア構成

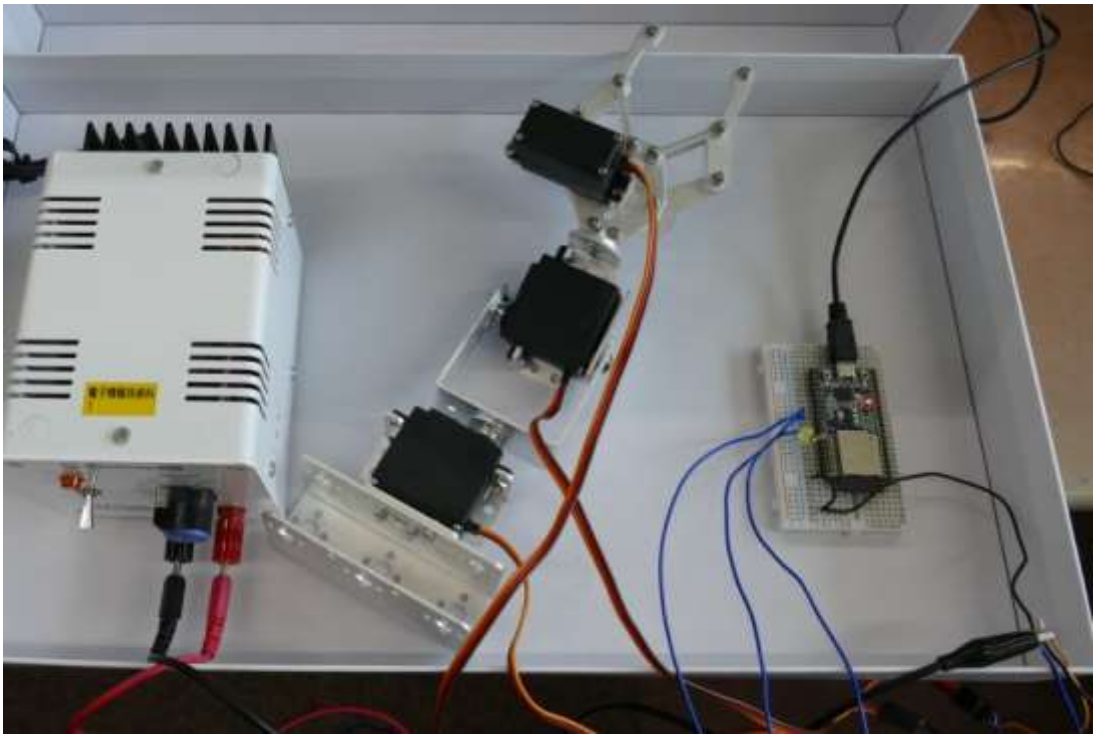
ESP32 で制御するものは、LED1個、サーボモーター3個としました。ほかにも制御対象があれば追加して構いません。



制御対象	ピン番号
サーボモーター1	IO16
サーボモーター2	IO17
サーボモーター3	IO18
LED	IO19

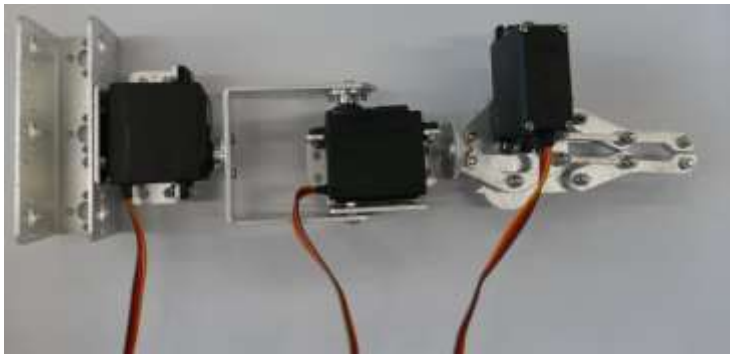
本来であれば、ESP32 の信号出力は 3.3V で動作しており、サーボモーターへの信号入力が5V でなければならないことがあります。場合によっては、信号が入らない可能性があるため 電圧レベル変換器 を利用した方がよいかもしれません。今回は、動作が行えたため利用しませんでした。

実際の配線は以下のようにしました。筆者の所属施設では、ロボットアームがありそれを活用しました。遠隔からこのロボットアームを動かしてみようと思います。本当であれば、ロボット型のマシン（サーボモーター多用のもの）が用意できれば面白くなると思います。



ロボットアームは撮影用に寝かせていますが、実験では机の上に立ち上がった状態で利用しています。

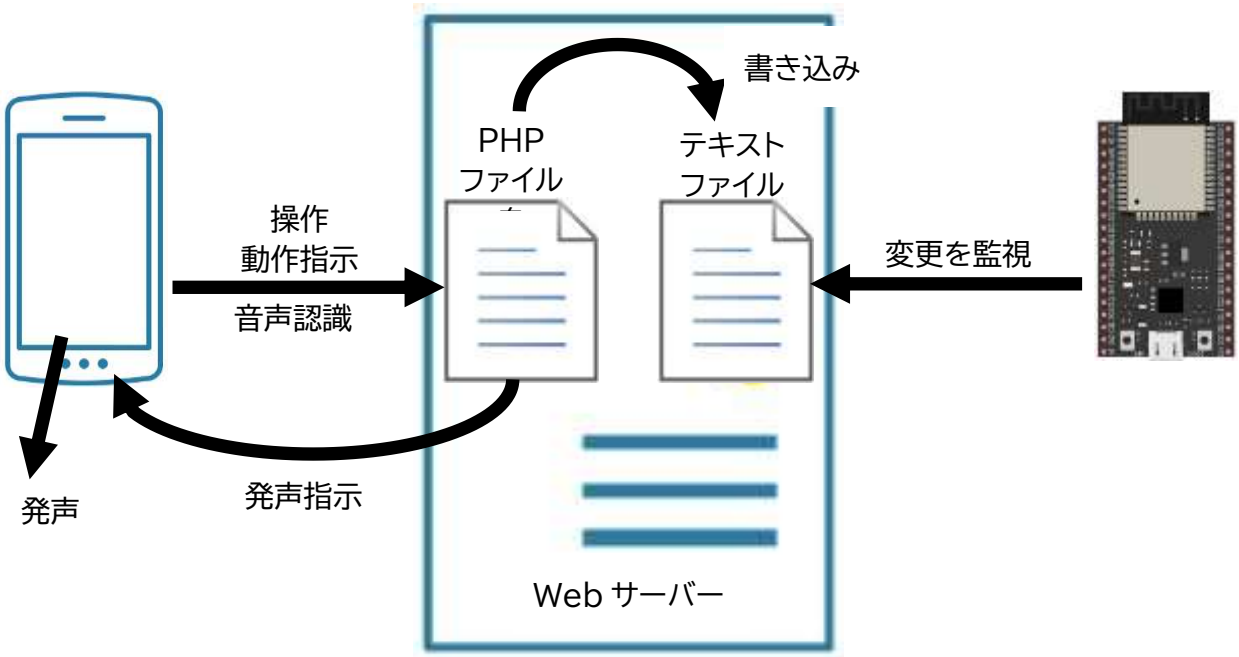
ロボットアームは、Amazon Japan から購入したものです。



IO16 へ　　IO17 へ　　IO18 へ

7.3　Web Speech API、PHP を用いたプログラム

全体の構成、動作の考え方を以下のようにしました。



以前の実習で作成したプログラムをほぼそのまま利用することができます。

作成するファイルは、以下の3ファイルとします（JavaScript の記述を別ファイルにして構いません）。

PHP ファイル （Web Speech API）	index.php
テキストファイル	datasource.txt
Arduino ファイル	keepwatch.ino

PHP ファイルには、操作端末からハードウェアの動作を声で受け付けます。その情報を GET 送信、ハードウェア制御する内容をテキストファイル、発声指示に書き込みを行います。

テキストファイル内には、ESP32 の動作指示内容が書き込まれます。

Arduino ファイルには、HTTP 通信を利用してテキストファイルを監視し続けます。タイミングとしては、10 秒でよいと思います。状況に応じて監視タイミングは調節します。

PHP ファイルを作成します。ファイル内には、Web Speech API の動作も併せて記述します。

動作パターンとして、LED2つ、サーボモーター6つとしました。もし、アイデアがあれば追加して構いません。

そこで音声認識パターンを以下のように検討してみました（サーボモーターの角度については、事前に1つずつ ESP32 で確認を行っています。サーボモーターを利用する際、個体差がよくあるので事前に確認するようにしてください）。

電気を点けて	LED オン（IO19）
電気を消して	LED オフ（IO19）
右向いて	IO16 を 180 度
左向いて	IO16 を 80 度　　元位置 130
はさみを開いて	IO18 を 90 度
はさみを閉じて	IO18 を 135 度
お辞儀して	IO17 を 30 度　　元位置 90
元に戻って	すべての位置情報を元の状態に戻す

音声認識の際に、どのように認識されるか事前に確認する必要があります。Web Speech API 実習の際に作成した rectest01.html を利用し、認識情報を確認します。

筆者の環境において、操作端末（Android タブレット）を用いて認識した音声情報を確認すると、以下のようになりました。

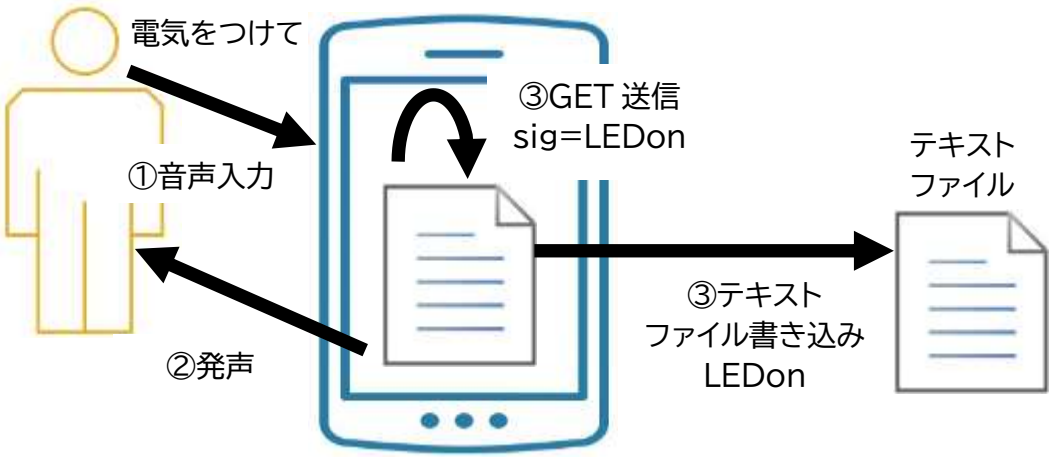
これは利用者の環境において多少違いがあるかもしれません。プログラム作成前に何度か試し、必ず確認しておいてください。

この結果を踏まえ、音声認識情報、GET メソッド送信する情報を以下とします。

音声認識情報	GET 送信情報
電気をつけて	sig=LEDon
電気を消して	sig=LEDoff
右向いて	sig=turnright
左向いて	sig=turnleft
ハサミを開いて	sig=openscissor
ハサミを閉じて	sig=closescissor
お辞儀して	sig=bow
元に戻って	sig=returnoriginal

テキストファイルに書き込む内容は、GET メソッド送信時の値のみとします。

再度確認ですが、音声入力後、以下のような動きになります



テキストファイルを作成し、その中に制御情報を書き込む準備をします。中身は、空で構いません。

```
# touch /var/www/html/datasource.txt
```

書き込み対象ファイルの所有者が apache ユーザーでなければ、書き込みができません。所有者を変更します。

```
# chown apache:apache /var/www/html/datasource.txt
```

JavaScript を用いて、Web Speech API を実行、認識した情報を GET 送信、PHP で GET 受信後にテキストファイルに書き込むプログラムを作成します。

ここでは、まず LED 点灯、消灯のみを確認します。

```
# vi /var/www/html/index.php
```

```
<!DOCTYPE html>
<html lang=”ja”>
<head>
  <meta charset=”UTF-8”>
  <script>
// 音声合成 API の使用
 let speech = new SpeechSynthesisUtterance();
 speech.lang = ”ja-JP”;
// 音声認識 API の使用
 let speechrec = new webkitSpeechRecognition();
 speechrec.lang = ”ja-JP”;

// 認識した音声情報を処理
 speechrec.onresult = function(event) {
   let results = event.results[0][0]. transcript;
   let speechstr = ””;                < 利用者に発声する内容を格納
   let getsignal = ”?sig=”;           < 転送する GET 情報を格納

   if (results == ””) {
       location.reload();
   } else if (results.indexOf(”電気をつけて”) >= 0 ) {
       getsignal += ”LEDon”;
       speechstr = ”電気をつけます”;
   } else if (results.indexOf(”電気を消して”) >= 0 ) {
       getsignal += ”LEDoff”;
       speechstr = ”電気を消します”;
   }
   speech.text = speechstr;
   speechSynthesis.speak(speech);      < 発声を指示

   location.href = getsignal;                < GET メソッド送信
}

// 音声認識開始
function get_send() {
    speechrec.start();
}
  </script>
</head>
```

次のページに続きます。

JavaScript では、以下により文字列を検索することができます。

```
検索対象文字列.indexOf(検索文字列)
```

戻り値として、検索文字列が検索対象文字列内にあれば、ヒットした位置情報を　0以上　の値で返します。

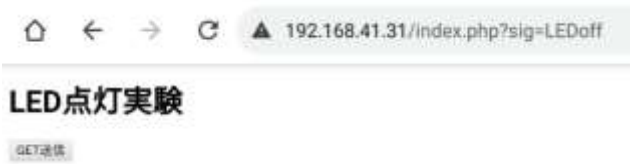
もし検索できなければ、-1 を返すことになっています。

```
<body>
<h1> LED 点灯実験 </h1>
<button onclick="get_send()">　GET 送信　</button>

<?php
  $filepath = "datasource.txt";
  $espdata = file_get_contents($filepath);　　< ファイルの中身を読み出し

  if (isset($_GET['sig'])) {　　< GET メソッド sig が存在すれば
      $espdata = $_GET['sig'];　< sig の値を espdata に格納
  } else {
      $espdata = "";　　< なければ、中身を消す
  }

  file_put_contents($filepath, $espdata);　　< ファイルに書き込み
?>

</body>
</html>
```

操作端末から Web サーバーに接続し、GET 送信 ボタンを選択。電気をつけて　としゃべりかけた直後の状態です。

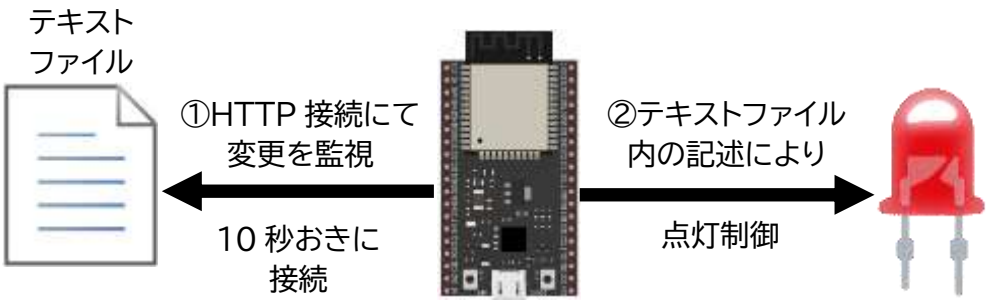
下記のように、GET メソッド送信が行われます。

同様に、電気を消して　としゃべりかけた直後の状態です。



7.4　LED 点灯 ESP32 制御プログラム

ESP 側の動作は、以下のようにします。



ESP32 の HTTP 接続に関するサンプルファイルが、ファイル ＞ スケッチ例 ＞ WiFi の中にあるので、参考にするとよいでしょう。

LED 制御部分のみを実行するプログラムは、以下のようになります。ファイル名は、keepwatch.ino とします。

```
#include <WiFi.h>
#include <HTTPClient.h>

int LEDPin = 19;              // LED のピン番号指定

// 接続するアクセスポイント指定情報
const char SSID[] = "speechapidemo";
const char PASSWORD[] = "polytechtest";

// IP 関連設定を行う
// ゲートウェイ、DNS はなくてもよいのですが、設定時にうまくいかないため設定
IPAddress ip(192, 168, 41, 32);                    // IP アドレス指定
IPAddress gateway(192,168, 42, 1);                 // ゲートウェイ指定
IPAddress subnet(255, 255, 255, 0);                // サブネットマスク指定
IPAddress DNS(8, 8, 4, 4);                         // DNS 指定

void setup() {
  Serial.begin(115200);
  pinMode(LEDPin, OUTPUT);              // LED のピン番号、出力指定

  WiFi.config(ip, gateway, subnet, DNS);           // IP アドレスなど設定
  delay(100);

  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(SSID, PASSWORD);                      // アクセスポイント接続
```

```
  while (WiFi.status() != WL_CONNECTED) {     // 接続が完了するまで待機
    delay(200);
    Serial.print(".");
  }
  Serial.println("WiFi connected");
}

// 監視対象を指定します。接続先に応じて、変更が必要になります
const char URL[] = "http://192.168.41.31/datasource.txt";

void loop() {                    // HTTP 接続実行
  HTTPClient http;
  http.begin(URL);
  int httpCode = http.GET();

  Serial.println("Response: " + httpCode);

  // 接続できたら、テキストファイル内の文字列を取得
  if (httpCode == HTTP_CODE_OK) {
    String bodydata = http.getString();
    Serial.print("Response Body: ");

    // 取得した文字列により制御
    if ( bodydata.indexOf("LEDon") >=0 ) {
      digitalWrite(LEDPin, HIGH);
      Serial.println("ONONON");
    } else if ( bodydata.indexOf("LEDoff") >=0 ) {
      digitalWrite(LEDPin, LOW);
      Serial.println("OFFFFFFF");
    } else {
      digitalWrite(LEDPin, LOW);
      Serial.println("ALLOFF");
    }
   // うまく動作しない時の確認用として
    Serial.println(bodydata);
  }

  // 10 秒おきに制御用ファイルを見に行く
  // 好きな時間に変更してもよい
  delay(10000);
}
```

Serial.println を多用していますが、きちんと動作するまでは確認が必要になります。動作確認が取れたようであれば削除して構いません。

7.5　サーボモーターの音声制御

LED 動作の確認ができれば、後は事前に確認したルールに基づいて、PHP ファイル、Arduino ファイルに追記するだけです。

PHP ファイルには、音声認識した情報により GET 送信を追加するだけになります。少し内容が長くなるので変更になる個所のみを確認します。先ほど作成したファイルに追記します。

```
# vi /var/www/html/index.php
```

```
// 認識した音声情報を処理
 speechrec.onresult = function(event) {
   let results = event.results[0][0]. transcript;
   let speechstr = "";                  < 利用者に発声する内容を格納
   let getsignal = "?sig=";             < 転送する GET 情報を格納

   if (results == "") {
       location.reload();
   } else if (results.indexOf("電気をつけて") >= 0 ) {
       getsignal += "LEDon";
       speechstr = "電気をつけます";
   } else if (results.indexOf("電気を消して") >= 0 ) {
       getsignal += "LEDoff";
       speechstr = "電気を消します";
   } else if (results.indexOf("右向いて") >= 0 ) {          < 以降、追記
       getsignal += "turnright";
   } else if (results.indexOf("左向いて") >= 0 ) {
       getsignal += "turnleft";
   } else if (results.indexOf("ハサミを開いて") >= 0 ) {
       getsignal += "openscissor";
   } else if (results.indexOf("ハサミを閉じて") >= 0 ) {
       getsignal += "closescissor";
   } else if (results.indexOf("お辞儀して") >= 0 ) {
       getsignal += "bow";
   } else if (results.indexOf("元に戻って") >= 0 ) {
       getsignal += "returnoriginal";
   }
   speech.text = speechstr;
   speechSynthesis.speak(speech);       < 発声を指示

   location.href = getsignal;                < GET メソッド送信
}
```

ESP32 音声制御

ESP32 を制御する Arduino ファイルもサーボモーターに関する情報を追記します。少し長くなりますが、こちらは全体で確認します。ファイルは、先ほど作成した keepwatch.ino です。

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ESP32Servo.h>          // サーボモーター用ライブラリ利用宣言

// サーボモーター用の初期設定
Servo servoDo1, servoDo2, servoDo3;
int servoPin1 = 16;          // 左右動作 サーボモーターのピン番号指定
int servoPin2 = 17;          // 前後動作 サーボモーターのピン番号指定
int servoPin3 = 18;          // ハサミ動作 サーボモーターのピン番号指定

// LED の初期設定
int LEDPin = 19;             // LED のピン番号指定

// 接続するアクセスポイント指定情報
const char SSID[] = "speechapidemo";
const char PASSWORD[] = "polytechtest";

// IP 関連設定を行う
// ゲートウェイ、DNS はなくてもよいのですが、設定時にうまくいかないため設定
IPAddress ip(192, 168, 41, 32);                    // IP アドレス指定
IPAddress gateway(192,168, 42, 1);                 // ゲートウェイ指定
IPAddress subnet(255, 255, 255, 0);                // サブネットマスク指定
IPAddress DNS(8, 8, 4, 4);                         // DNS 指定

void setup() {
  Serial.begin(115200);

  // サーボモーター用初期設定
  servoDo1.setPeriodHertz(50);
  servoDo1.attach(servoPin1);
  servoDo1.write(90);
  servoDo2.setPeriodHertz(50);
  servoDo2.attach(servoPin2);
  servoDo2.write(90);
  servoDo3.setPeriodHertz(50);
  servoDo3.attach(servoPin3);
  servoDo3.write(90);

  pinMode(LEDPin, OUTPUT);     // LED のピン番号、出力指定

  WiFi.config(ip, gateway, subnet, DNS);          // IP アドレス設定
  delay(100);
```

```
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(SSID, PASSWORD);                    // アクセスポイントに接続
  Serial.print("WiFi connecting");

  while (WiFi.status() != WL_CONNECTED) {        // 接続が完了するまで待機
    delay(200);
    Serial.print(".");
  }
  Serial.println("WiFi connected");
}

// 監視対象を指定します。接続先に応じて、変更が必要になります
const char URL[] = "http://192.168.41.31/datasource.txt";

void loop() {
  HTTPClient http;
  http.begin(URL);
  int httpCode = http.GET();

  Serial.print("Response:" + httpCode);
  Serial.println();

  // 接続できたら、テキストファイル内の文字列を取得
  if (httpCode == HTTP_CODE_OK) {
    String bodydata = http.getString();
    Serial.print("Response Body: ");

    // 取得した文字列により制御
    if ( bodydata.indexOf("LEDon") >=0 ) {
      digitalWrite(LEDPin, HIGH);
      Serial.println("ONONON");
    } else if ( bodydata.indexOf("LEDoff") >=0 ) {
      digitalWrite(LEDPin, LOW);
      Serial.println("OFFFFFFF");
    } else if ( bodydata.indexOf("turnright") >=0 ) {
      servoDo1.write(180);                       // IO16 を 180 度
      Serial.println("turnright");
      delay(1000);
    } else if ( bodydata.indexOf("turnleft") >=0 ) {
      servoDo1.write(80);                        // IO16 を 80 度
      Serial.println("turnleft");
      delay(1000);
    } else if ( bodydata.indexOf("openscissor") >=0 ) {
```

```
        servoDo3.write(90);                          // IO18 を 90 度
        Serial.println("openscissor");
        delay(1000);
      } else if ( bodydata.indexOf("closescissor") >=0 ) {
        servoDo3.write(135);                         // IO18 を 135 度
        Serial.println("closescissor ");
        delay(1000);
      } else if ( bodydata.indexOf("bow") >=0 ) {
        servoDo2.write(30);                          // IO17 を 30 度
        Serial.println("bow");
        delay(1000);
      } else if ( bodydata.indexOf("returnoriginal") >=0 ) {
        servoDo1.write(130);                         // IO16 を 130 度
        delay(1000);
        servoDo3.write(135);                         // IO18 を 135 度
        delay(1000);
        servoDo2.write(90);                          // IO17 を 90 度
        Serial.println("turnoriginal");
        delay(1000);
      } else {                                       //  適切でなければ、すべて元に戻す
        digitalWrite(LEDPin, LOW);
        Serial.println("ALLOFF");
        servoDo1.write(130);
        delay(1000);
        servoDo3.write(135);
        delay(1000);
        servoDo2.write(90);
        Serial.println("ALLOFF");
        delay(1000);
      }
     // うまく動作しない時の確認用として
     Serial.println(bodydata);
   }

   // 10 秒おきに制御用ファイルを見に行く
   // 好きな時間に変更してもよい
   delay(10000);
}
```

先ほどと同じく、Serial.println の記述がありますが、動作確認が取れれば削除して構いません。

以上が全体の流れになります。

7.6　今後について

ここまでの内容により、ESP32 に様々なハードウェアを接続し、音声により制御する方法について理解できたのではないでしょうか？　この後は、会話ルール、制御対象を増やしていけばたくさんのことができるようになります。この考え方は、市販のスマートスピーカーにオリジナル機能を作成する時の助けにもなります。

筆者が実際に作成したものは、以下のようなものです

- 100V 制御し、部屋の電灯制御、扇風機制御（リレーを利用したオンオフのみ）
- アームを利用して物を掴むまでの一連の処理
- ロボットにダンスをさせる　　などです

これ以外にも、Web ページの意匠が気になるようであれば、スタイルシートを駆使しデザイン性の優れたインターフェイスを作成することもよいでしょう。他にも操作端末用のオリジナル Android アプリを制作してもよいかもしれません。

ネットワーク、セキュリティなど諸条件を満たす必要がありますが、レンタルサーバー、AWS、GCP などのクラウドサービスを利用し、インターネット上に今回作成した Web サーバーを構築します。ESP32、操作端末をインターネット接続すると、どこからでも ESP32 のハードウェア制御が可能になります。

ぜひ、アイデアを考え、考えたことが実現できるよう色々なことに取り組んでみてください。

8　あとがき

現在、ESP32 の通信機能を活用した IoT はたくさん考えられています。無線 LAN、Bluetooth を利用した制御には、定評があります（また、ESP32 は小型化、M5Stack、obniz など利用しやすい製品が次々登場しています。これら活用して IoT 機器を作成してみることもよいと思っています。ぜひ取り組んでみてください）。この部分をもっと楽しいこと、制作物に活かせないか考えた末に出来たものが、この教材の音声認識です。筆者もスマートスピーカーを始めて利用した際には、音声認識の精度の高さに驚きました。この音声認識を制作する部分は無理でもうまく利用することはできないかと思っていました。それが、Web Speech API でした。オリジナルスマートスピーカーを作るよりもかなりハードルが低かったからです。

今回の実習を踏まえクラウドサービスを利用した環境で、同じことをやってみてはいかがでしょうか。この実習を通じ、大まかなシステムの動き、ネットワーク構成などが見えてきたと思います。

この実習書の作成以前では、Web Speech API 音声認識は Web サーバーなしでは動作しませんでした。しかし、最近の新しい Chrome ブラウザでは、Web Speech API の記述のある HTML ファイルがローカル環境にあっても動作するようです。もし、簡単に音声認識、合成の確認だけをしたければ、HTML ファイルのみを作成して実行できるかもしれません。

筆者としては、全体の理解を深めるためネットワーク、サーバーなども含めて作業を行った方が後につながるような気がしています。

ハードウェア制御について、ESP32 から Web サーバーのテキストファイルを監視する形にしましたが、タイミングによりどうしても動作を遅く感じるかもしれません。最大で 10 秒間の間があるためです。この間隔を短くすればよいのですが、ESP32 に負担が大きいかもしれません。

この実習では触れていませんが、実習が終了した後には、MQTT（Message Queue Telemetry Transport）を利用した連携について取り組んでもよいかもしれません。MQTT は、今後の IoT における重要な要素になる可能性があるためです。実習での動作遅延を解消できる可能性があります。ESP32 におけるプログラムもそんなに難しくありません。これに併せて、ESP32 には、ディープスリープ機能があります。これを組み合わせて、省電力機能を搭載した IoT を作成してもよいですね。

この実習を通じて、音声認識を利用した仕組みの一部分を理解できたことを願います。

読者の皆さんが、これからも IoT 関連についてさらに理解を深め、自分の考えたアイデアを実現できることを期待しています。

9　参考文献

※1 総務省トップ ＞ 政策 ＞ 白書 ＞ 令和元年版 ＞ レイヤー別にみる市場動向 より
https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r01/html/nd112130.html

※2　Espressif Systems Support Documents Technical Documents
https://www.espressif.com/en/support/documents/technical-documents

※3　ESP32-WROOM-32
https://ht-deko.com/arduino/pic/esp32_devkitc_pinout_01.png

※4【CentOS8】日本語入力ができないときの対処方法
https://qiita.com/LemonLeaf/items/68a15edf87c7f402b1f1

※5　Web Speech API
https://developer.mozilla.org/ja/docs/Web/API/Web_Speech_API

※6 SpeechSynthesis
https://developer.mozilla.org/ja/docs/Web/API/SpeechSynthesis

※7 SpeechRecognition
https://developer.mozilla.org/ja/docs/Web/API/SpeechRecognition

※8 Web Speech API Draft Community Group Report, 2 July 2020
https://wicg.github.io/speech-api/#examples-recognition