第３章　ＳＮＳを利用したＩｏＴアプリケーション開発

３．１　製造現場におけるＳＮＳ活用

ＳＮＳ（Social Networking Service）には様々なものがあり，特定の人物との会話や不特定多数の相手との連絡手段，飲食店や商業施設のＰＲなど様々な活用がされています．

ＳＮＳの中にはＡＰＩ(Application Programable Interface)が公開されているものもあり，他のアプリケーションと連携させることが容易となっているものもあります．このようにＡＰＩを活用することで生産現場におけるＳＮＳ活用の幅が広がる可能性があります．

例として製造装置の遠隔制御・監視方法を考えてみます．遠隔地から特定の制御装置には制御用コンピュータ（ＰＬＣ）にアクセスする必要があります．制御用コンピュータはネットワークからソケットを使用した特定のコマンドを受信することで内部レジスタや接点情報を取得することができます．　この場合，コマンド送受信をするためのアプリケーションがインストールされているコンピュータしかアクセスできないためセキュリティ面では強固である反面，インターフェース設計など他のプログラム言語などの知識を要します．

一方，遠隔制御・監視手段としてＳＮＳを活用すると操作方法などは普段の使用方法と変わらないため時間はかからないですし，外部ネットワークに接続するためローカルエリア内の端末以外の端末からアクセスすることも可能です．またスマートフォンの通知機能と併用すると緊急事態における端末へのサウンド、バイブレーションを活用した即時を行うことができるため迅速な対応もしやすくなります．

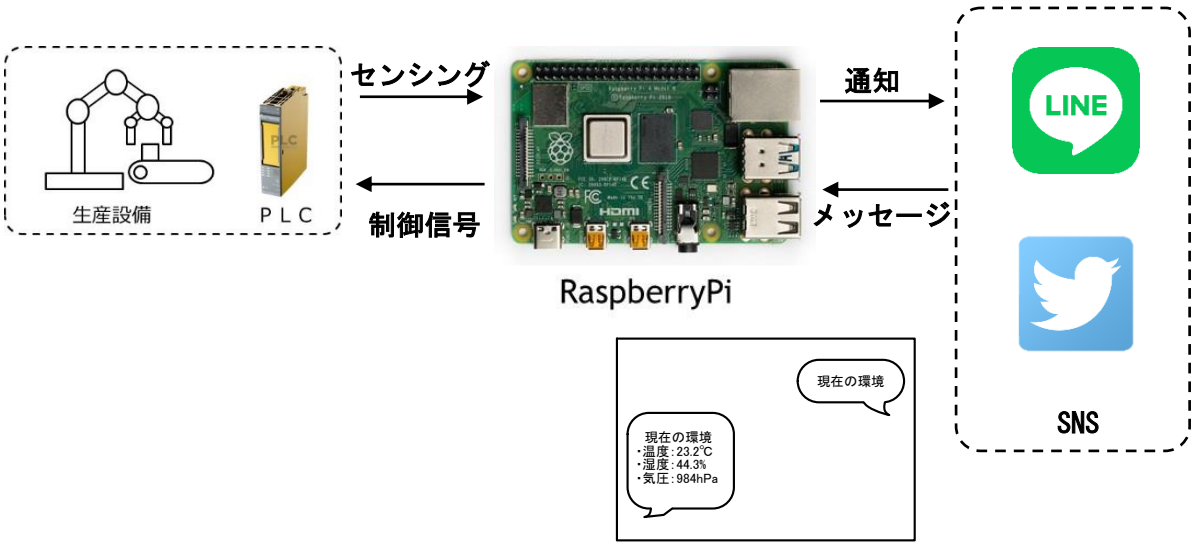


図３．１　ＳＮＳ活用のイメージ

本実習で使用する SNS は LINE®です．LINE は国内で使用されている SNS として広く知られているだけでなく，LINE bot という仕組みが公開されており API を経由して誰でも簡単に自動コミュニケーションツールを活用することができます．LINE bot を使用することでメッセージやデータ通知を利用して自動的に行うことができます．

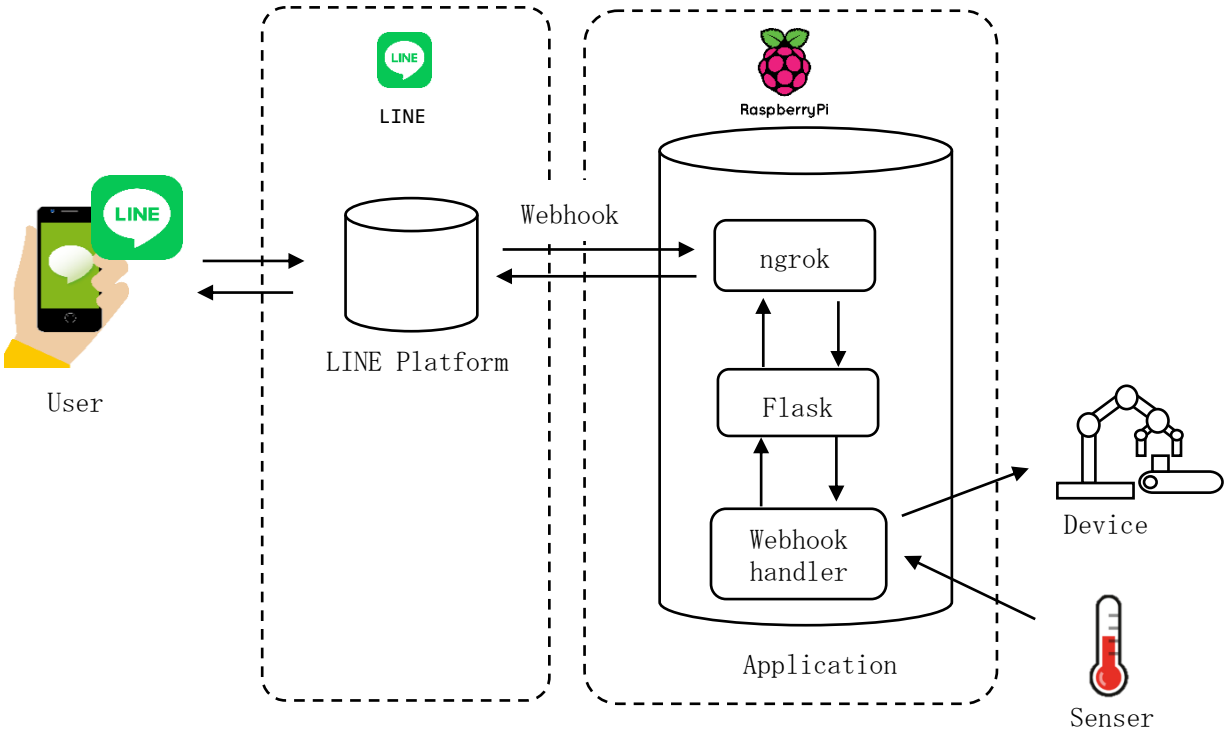


図３．２　LINE bot の仕組み

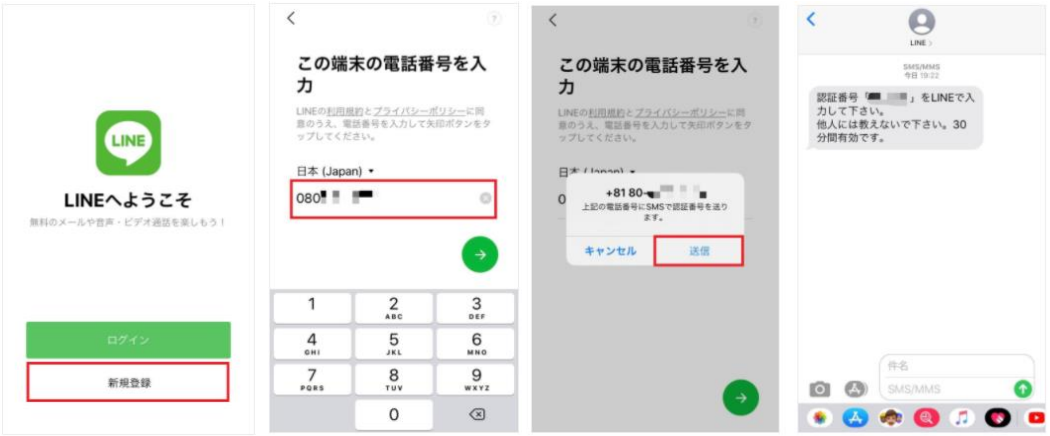
ユーザーが LINE でメッセージを送ると webhook に登録されたアドレスに HTTPS リクエストを送ります．外部公開されている URL をローカル IP に変換し(ngrok)，web フレームワーク(Flask)のコールバックルーチンを呼び出します．コールバックルーチンでは LINE が提供している MessagingAPI を利用して送信されたメッセージをアプリケーション上で使用することができます．メッセージ内容によってデバイス制御やセンサ取得ができます。

Webhook: アプリケーションの更新情報を他のアプリケーションへリアルタイム提供する仕組み
ngrok:通常はローカル環境でアクセスできる URL を外部公開するサービス
flask:Python で扱うことができる軽量 Web フレームワーク
Webhookhandler:Webhook からのイベントを受信したときに実行する処理(コールバック)

３．２　ＬＩＮＥ　ＡＰＩの環境構築

(1)LINE アカウント作成

①　②　③　④



⑤　⑥　⑦　⑧

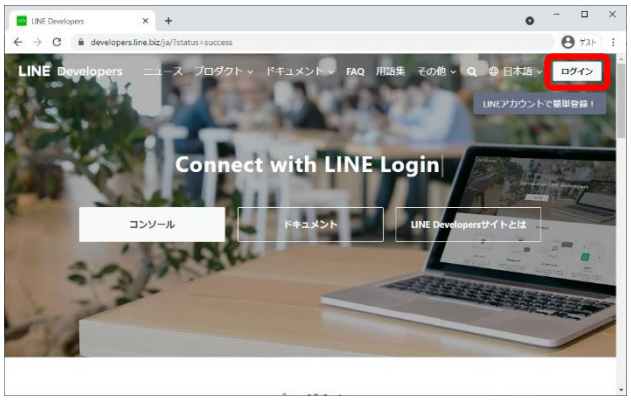
⑨　⑩　⑪　⑫

図３．３　ＬＩＮＥアカウントの取得

(2)LINE Developer のログイン

URL:https://developers.line.biz/ja/

①

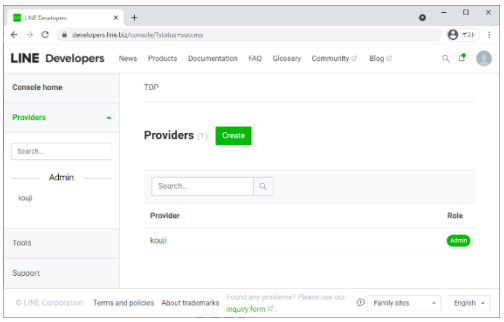


②

③

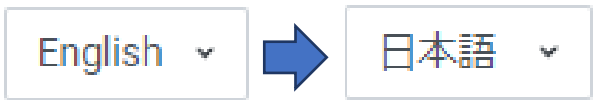
④

⑤



⑥

日本語表記に変更



⑦

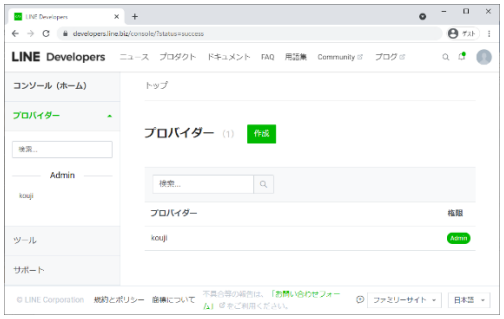


図３．４　LINE Developer へのログイン

(3)プロバイダー作成

LINE プラットフォームを通じてサービスを提供する個人や企業を登録します.

図３．５　新規プロバイダー作成

(4)MessagingAPI チャネル作成

MessagingAPI とは，LINE プラットフォームからのメッセージの通知もしくはプログラムコードで生成したメッセージを LINE プラットロームに送信する API のことです.

Messaging API で提供されるサービスには下記のものがあります.

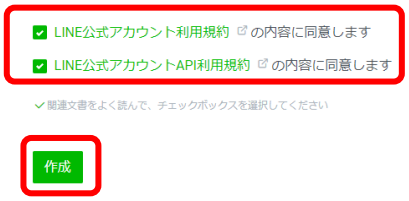
① 応答メッセージを送る
② プッシュメッセージを送る
③ その他さまざまなタイプのメッセージ（テキスト、スタンプ、画像、動画、音声、位置情報、イメージマップ、テンプレート）
④ ユーザーが送ったコンテンツを取得する
⑤ ユーザープロフィールを取得する
⑥ グループチャットに参加する
⑦ リッチメニューを使う
⑧ ビーコンを使う
⑨ アカウント連携を使う
⑩ 送信メッセージ数を取得する

MessagingAPI を使用するには下記の順に MessagingAPI チャネルを作成します．

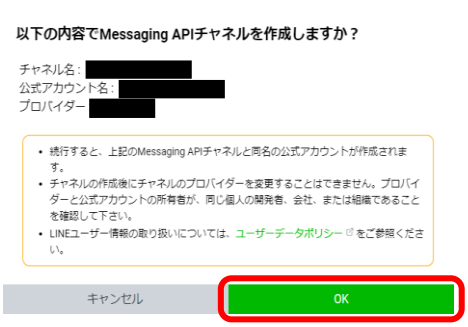
①

チャネル名（必須）大業種（必須)、小業種（必須)、利用規約への同意を記入する．

②



③



④

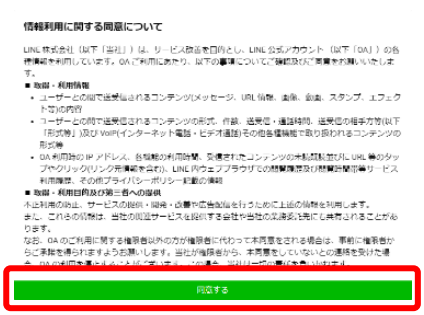


図３．６　MessagingAPI チャネル作成

(5)環境変数にチャネルアクセストークン，シークレットトークンを設定する．

LINE Developer のシークレットトークンとチャネルアクセストークンを環境変数へ代入します．各トークンは作成したチャンネルの基本設定および Messaging API 設定の画面で確認できます．

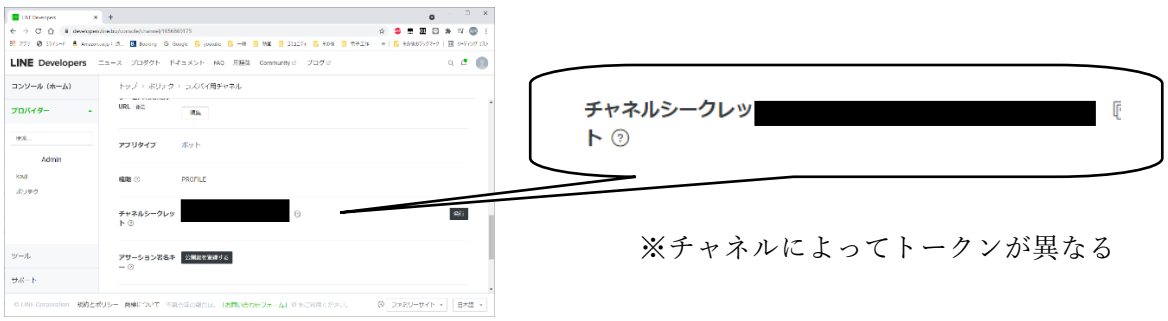


図３．７　シークレットトークンの確認

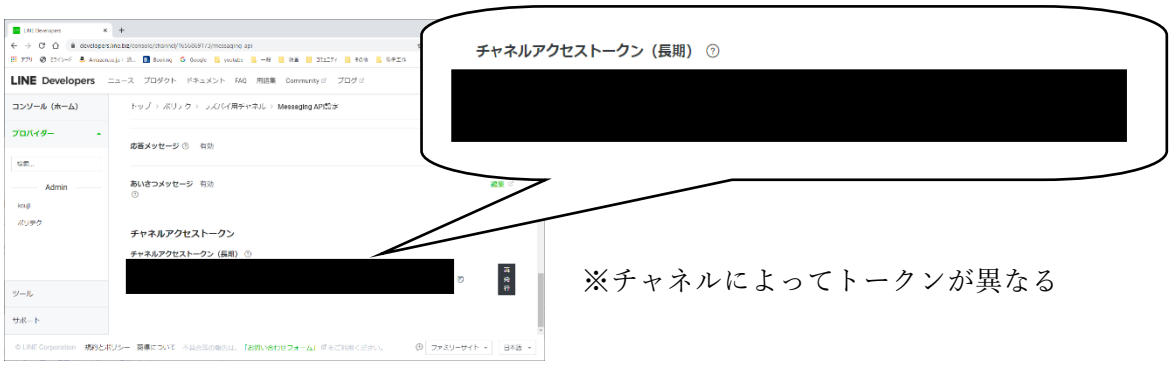


図３．８　チャネルアクセストークンの確認

①環境変数へ代入

```
$ nano ~/.bashrc
```

②.bashrc の末尾に追加する

図３．７と図３．８で確認した各トークンをコピー&ペーストします.

```
#シークレットトークン
export LINE_CHANNEL_SECRET=********************
#チャネルアクセストークン
export LINE_CHANNEL_ACCESS_TOKEN=**********************
```

③もし社内プロキシを採用しているネットワークの場合は同一の.bashrc に下記の記述を追加します.

```
export HTTP_PROXY=http://<user>:<password>@<proxyserver>:<port>
export HTTPS_PROXY=https://<user>:<password>@<proxyserver>:<port>
```

ポリテクセンター山梨ではプロキシサーバを採用しているため下記の記述のように末尾に記載します.

```
export HTTP_PROXY=http://10.0.0.2:15080
export HTTPS_PROXY=https://10.0.0.2:15080
```

(6)LINE API のインストール

```
$sudo pip3 --proxy=http://10.0.0.2:15080 install line-bot-sdk
```

(7)ngrok ダウンロードとインストール

ngrok(エングロック)とは，ローカル環境で実行している WEB アプリケーション(http,thhps,TCP)を外部公開するサービスです．ngrok には有償版と無償版があります．

有償版：プランによってひと月当たりの料金が変わります．公開される URL を自由に設定できる（つまり固定化できる）などメリットがあります．
無償版：料金がかからない代わりに，公開される URL を固定化することができません．ngrok を起動させるたびに URL がランダムに変わります．また，ダウンロードの際にユーザー登録しないと公開 URL への接続に時間制限が設けられます（２時間程度）

本実習では ngrok 無償版を使用します．必要に応じて有償版にアップグレードできます．

① 下記の URL からダウンロード・展開する

```
$wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-arm.zip
$unzip ngrok-stable-linux-arm.zip
```

②ダウンロードした展開してラズベリーパイへコピー
コピー先： /usr/local/bin/

```
$sudo  mv ./ngrok  /usr/local/bin/
```

(9)ngrok 実行（本実習ではポート番号を 8080 とします）

```
$ngrok http 8080
```

ngrok によって外部公開される URL(https)を確認します．下記のように外部公開された URL を LINE Developer の Webhook URL に登録します．

TeraTarm を使用している場合，下記の網掛けになっている URL をドラックすることでクリップボードにコピーできます．

```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Session Expires               1 hour, 57 minutes
Version                       2.3.40
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http:// ******************.ngrok.io -> http://localhost:8080
Forwarding                    https://******************.ngrok.io -> http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

外部公開される URL(https)をコピーする！

※この URL は ngrok（無料版）が起動するたびにランダムで与えられます.

図３．９　ngrok 起動画面

(10)Webhook URL に登録する

図３．１０　WebhookURL の設定

(11)応答メッセージを設定する

初期状態では LINEbot が受信したメッセージに対して自動返信する設定となっているため無効にします.

「応答メッセージ」－「編集」

「あいさつメッセージ」をオフ

「応答メッセージ」をオフ

図３．１１　応答メッセージ設定

(12)メッセージイベントアプリをコピー

コピー元 z:\work\LINE

コピー先 /home/pi/work/

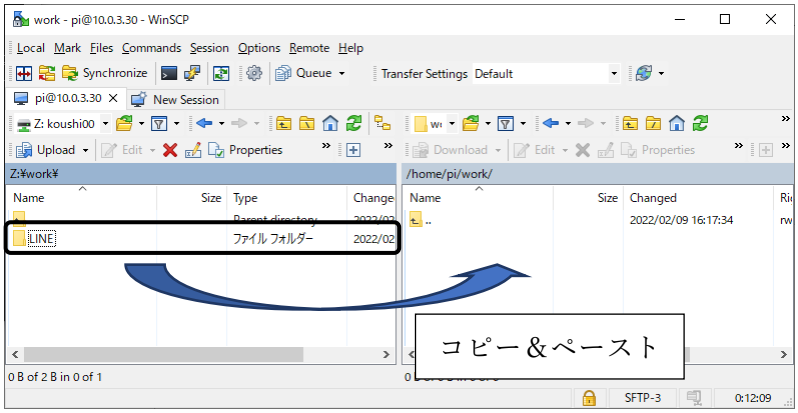


図３．１２　メッセージイベントアプリのコピー

(13)ターミナル画面を複製してメッセージイベントアプリを起動させる

①

[File]-[Duplicate session]をクリック

図３．１３　ターミナルの複製

②

```
$ cd ~/work/LINE
$ python3 ./messageEvent.py
```

(14)QR コード読み取り

スマートフォンもしくはタブレット PC の LINEbot の QR コードを読み取ります

図３．１４　チャネル登録

(15)作成したチャネルにメッセージを送る

メッセージを送ると同じメッセージが送り返されることを確認します。

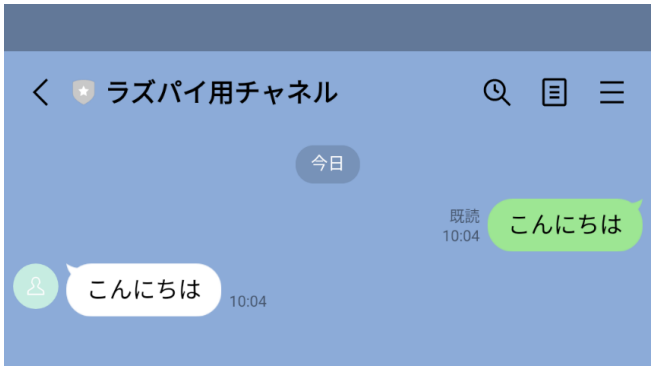


図３．１５　メッセージ送信

３．３　ＬＩＮＥ　ＡＰＩ（メッセージイベント）

サンプルコードを例にそれぞれの LINE API を解説します.

(1)LINE bot ライブラリのインポート

LINE API を使用するには下記のように LINE bot ライブラリをインポートします．非常に多くのライブラリやモジュール（クラス）が存在します．LINEbot ライブラリの中の２つのライブラリ(LineBotApi, WebhookHandler)をインポートします.

```
from linebot import (
    LineBotApi, WebhookHandler
)
```

(2)各種イベントモジュールのインポート

```
from linebot.models import (
    MessageEvent, TextMessage, TextSendMessage,
    ……
)
```

WebhookHanlder メソッドに必要なイベントモジュールは下記の通りです.

linebot.models:LINEbot の各種イベントを定義しているライブラリ

MessageEvent:メッセージを受信した際に通知されるイベント

TextMessage:メッセージオブジェクト

TextSendMessage:LINE プラットフォームにメッセージ送信する関数

(3)チャネルアクセストークンとチャネルシークレットトークンを読み込み

　チャンネルアクセストークンはあらかじめ環境変数に代入されているためコードのなかで読み込みします.

```
import os
……

#LINE チャネルシークレットトークン環境変数読み込み
channel_secret = os.getenv('LINE_CHANNEL_SECRET', None)

#LINE チャネルアクセストークン環境変数読み込み
channel_access_token = os.getenv('LINE_CHANNEL_ACCESS_TOKEN', None)
```

(4)LINE bot API のインスタンス生成

LINE bot API のインスタンスを生成するにはチャンネルアクセストークンを必要とします.

```
line_bot_api = LineBotApi(channel_access_token)
```

(5)WebhookHandler インスタンス生成

WebhookHandler のインスタンスを生成するにはチャネルシークレットトークンを必要とします.

```
handler = WebhookHandler(channel_secret)
```

(6)電子署名の検証ルーチン（コールバックルーチン）

リクエストが LINE プラットフォームから送られてきたものなのか検証するため，イベントの通知をする前に電子署名を検証します．検証するためには「X-Line-Signature」が含まれている HTTPS リクエストヘッダを取得します.

```
@app.route(“/callback”, methods=[‘POST’])
def callback():
    # HTTP リクエストのヘッダを取得
    signature = request.headers['X-Line-Signature']

    # HTTP リクエスト(POST)のボディを取得
    body = request.get_data(as_text=True)
    app.logger.info("Request body: " + body)

    # 署名を検証し、問題なければ@handler.add に定義している関数を呼び出す
    try:
        handler.handle(body, signature)
    except InvalidSignatureError:
        abort( 400 )

    return 'OK'
```

WebhookHandler メソッドの呼び出し（署名の検証も同時に行う）

(7)WebhookHandler メソッドの追加

自作関数を WebhookHandler メソッドに追加するには@handler.add を記述します.
handler.add には通知するイベント(MessageEvent)とメッセージオブジェクト(TextMessage)を引数として渡します．メッセージオブジェクトは自作関数の引数(event)に引き継がれます.

```
@handler.add(MessageEvent, message=TextMessage)
def handl_text_message(event):
    ……
```

自作関数(WebhookHandler)に登録

(8)LINE からのメッセージ取得

LINE からメッセージを取得するには HTTP リクエストボディから取得する必要があります．自作関数の引数(event)には HTTPS リクエストボディが代入されています．

```
@handler.add(MessageEvent, message=TextMessage)
def handl_text_message(event):
    #メッセージを取得
    text = event.message.text

    ……
```

LINE からのメッセージは文字列で取得できます．従って，これ以降の処理は通常の文字列処理として分岐や解析を行うことが可能です．

(9)LINE へのメッセージ送信

LINE に対してメッセージを送信するには LINEbotAPI における reply_message メソッドを使用します．reply_message メソッドには応答用トークンとメッセージ関数として TextSendMessage 関数を使用します．

```
@handler.add(MessageEvent, message=TextMessage)
def handl_text_message(event):

   ……

   #LINE にメッセージ送信
   line_bot_api.reply_message(
      event.reply_token,      #リクエスト応答用トークン
      TextSendMessage(text)  #送信メッセージ関数（メッセージをセット）
   )
```

送信メッセージは固定の文字列だけでなく，センサ情報を文字列に変換することで送信することができます．

```
@handler.add(MessageEvent, message=TextMessage)
def handl_text_message(event):
   #環境センサ(BME280)から温度,気圧,湿度データの取得(float 型)
   tmp,pre,hum = bme280.readData()

   #LINE にメッセージ送信
   line_bot_api.reply_message(
      event.reply_token,          #リクエスト応答用トークン
      TextSendMessage(f"{tmp:.1f}")   #温度データ(少数第 1 位)を文字列に変換しセットする
   )
```

(10)メッセージ送受信シーケンス

ユーザーが LINE にメッセージを送信しメッセージが送り返されるまでの流れを下記の図に示します．ユーザーがメッセージを送信すると LINE から Webhook を通じて ngrok で公開されている端末に HTTPS リクエストを送ります．Web フレームワークによってエンドポイントに指定されているコールバック関数が実行されます．コールバック関数では電子署名の検証と WebhookHandler の呼び出しを行い，”OK”を戻します．登録された WebhookHandler では受信メッセージの解析や周辺回路の制御を行い，必要に応じてメッセージを返します(line_bot_api.reply_message メソッド)．

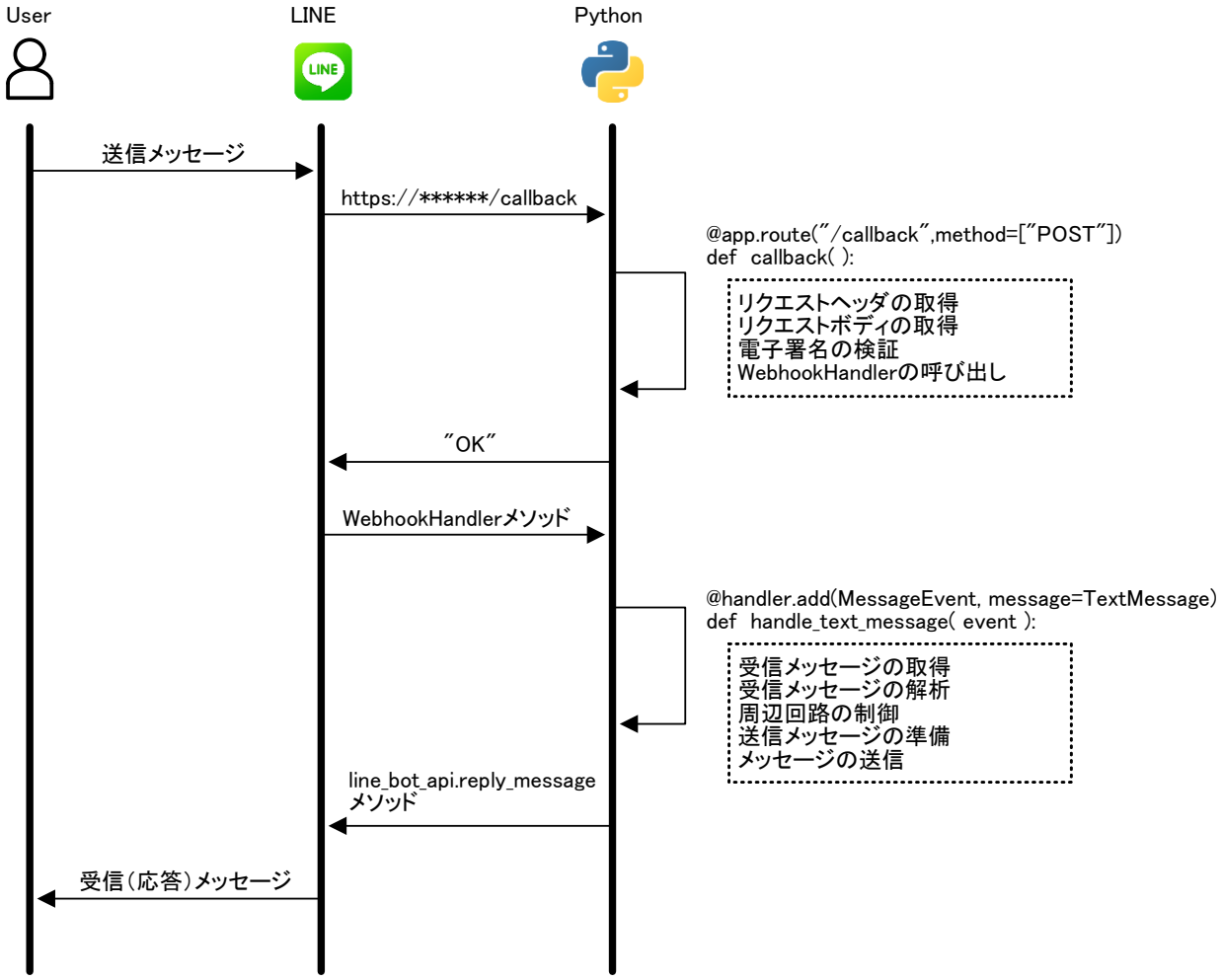


図３．１６　LINEbot におけるシーケンス図

３．４　練習問題（システム内部の温度取得）

(1)　仕様

ラズベリーパイ用 LINE チャネルに対して「温度を表示」とメッセージを表示すると「現在　○○℃です」と表示し，それ以外のメッセージを受信した場合「わかりません」と表示する LINEbot を作成しましょう．

(2)ファイル名とファイルパス

ファイル名：tempEvent.py

ファイルパス：/home/pi/work/LINE/

(3) 温度センサ

本課題ではラズベリーパイに接続されているサンハヤト製拡張 IO ボードを使用します．実装されている温度センサ TMP102 から温度を取得し，LINE メッセージに送信しましょう．

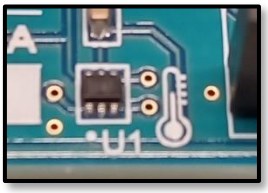


図３．１７　温度センサ(TMP102)

```
#温度センサライブラリのインポート
from tmp102 import TMP102

……

#インスタンス生成
tmp = TMP102()

#温度取得(float 型)
t=tmp.readTemperature()
```

(4)実行例

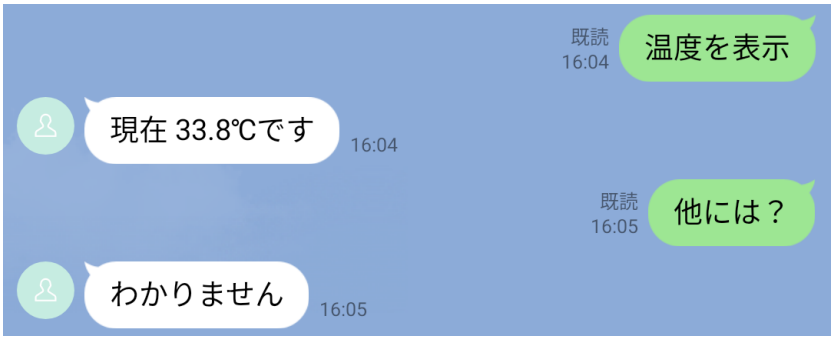


図３．１８　システム温度の問い合わせ例

３．５　搬送負荷装置ライブラリ

第１章で使用した MC プロトコルを使用し負荷装置の稼働状況を取得します．ただし MC プロトコルは要求伝文（パケット）が複雑であるため，LINEbot プログラムの中にすべて記述すると可読性が悪くなるうえ不具合発生時のデバッグに時間がかかります．

本実習ではあらかじめ搬送負荷装置の各種情報の取得と設定をするための専用ライブラリを用意しました．

(1)ライブラリインポート

```
from PLCCOM import PLCCOM
```

(2)ライブラリ解説

①インスタンス生成

関数名	PLCCOM(IPAddress, PORT)
引数	IPAddress: PLC の IP アドレス(str 型) PORT: PLC のポート番号(int 型)
戻り値	PLC と通信するインスタンス
概要	PLC と通信するインスタンスを生成する

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	…

②稼働状況の取得

関数名	read_machine_state()
引数	なし
戻り値	“STOP”　　: 停止中 “DRIVE”　　: 稼働中 “EMERGENCY” : 非常停止中 “RESET”　　: 原点復帰中
概要	搬送負荷装置の状態を取得する

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	
	#装置の状態を取得
	state = plc.read_machine_state()
	
	#もし機械が非常停止中だったら…
	if state==”EMERGENCY”:
	　……

③生産目標数の取得

関数名	read_product_target()
引数	なし
戻り値	生産目標数(int 型)
概要	生産目標数を取得する

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	
	#装置の生産目標数を取得
	ptget = plc.read_product_target()
	
	#現在の生産目標数の表示
	print(f“現在の目標数は{ptget}個です.”)

④生産数の取得

関数名	read_product()
引数	なし
戻り値	生産数(int 型)
概要	生産数を取得する．生産数とは良品の数と不良品の数を加算した値.

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	
	#装置の生産目標数を取得
	pt = plc.read_product()
	
	#現在の生産目標数の表示
	print(f“現在の生産数は{pt}個です.”)

⑤良品数の取得

関数名	read_good_product()
引数	なし
戻り値	良品の数(int 型)
概要	良品の数を取得する.

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	
	#装置の良品数を取得
	good_pt = plc.read_good_product()
	
	#現在の良品数の表示
	print(f“現在の良品数は{good_pt}個です.”)

⑥不良品数の取得

関数名	read_defective_product()
引数	なし
戻り値	良品の数(int 型)
概要	良品の数を取得する.

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	
	#装置の不良品を取得
	defective_pt = plc.read_defective_product()
	
	#現在の不良品の表示
	print(f“現在の良品数は{defective_pt}個です.”)

３．６　練習問題（搬送負荷装置における稼働状況の問い合わせ）

(1)　仕様

ラズベリーパイ用 LINE チャネルに対して「稼働状況」とメッセージを送信すると搬送負荷装置の稼働状況を「現在〇〇です」と表示し，それ以外のメッセージを受信した場合「わかりません」と表示する LINEbot を作成しましょう.

(2)ファイル名とファイルパス

ファイル名：machineStateEvent.py

ファイルパス：/home/pi/work/LINE/

(3)実行例

図３．１９　LINE による稼働状況の問い合わせ例

(4)記述例

```
@handler.add(MessageEvent, message=TextMessage)
def handle_text_message(event):
    #LINE から受信した文字を代入
    text = event.message.text

    if text=="稼働状況":

        #搬送負荷装置ライブラリのインスタンス生成
        plc=PLCCOM("???.???.???.???", ????)
        #稼働状況の取得
        state = plc.??????????????????()

        #機械の稼働状況を判断、メッセージをセット
        if state=="???????":  #もし state が”STOP”だったら…
                message = "現在停止中です"
        elif state=="???????":  #もし state が”DRIVE”だったら…
                message = "現在稼働中です"
        elif state=="???????":  #もし state が”EMERGENCY”だったら…
                message = "現在緊急停止中です"
        else:
                message = "システムエラー"

        #稼働状況を LINE で送り返す
        line_bot_api.reply_message(
                event.reply_token,              #リクエスト応答用トークン
                TextSendMessage(text=message)#メッセージ送信
        )

    else:
        #「わかりません」を LINE で送り返す
        line_bot_api.reply_message(
                event.reply_token,              #リクエスト応答用トークン
                TextSendMessage("わかりません")        #メッセージ送信
        )
```

(5)追加仕様

「稼働状況」のメッセージを送信すると搬送負荷装置の稼働状況を「現在○○です」と表示するとともに生産数，良品数，不良品数を表示するように改良しましょう．

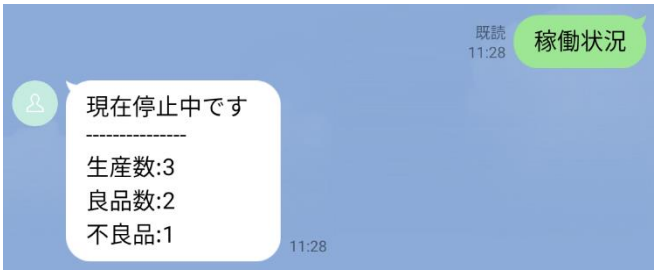


図３．２０　稼働状況と生産状況の問い合わせ例

３．７　通知機能の実装

これまでの LINE を使用した稼働状況の取得では，ユーザーが問い合わせをすることで LINEbot が応答する形式となっていました．ところがシステムエラーや何らかの要因により設備が緊急停止する状況が発生した場合は，問い合わせを待つことなく状況を通知することで手早くエラー要因を特定することができます．

LINEbotAPI にはメッセージを通知するプッシュ通知機能が実装されています．メッセージイベントのプログラムコードと比較して少ないコード量で実装できます．

(1)ユーザーＩＤの確認

プッシュ通知機能を使用するには作成したチャネルのユーザーＩＤを確認する必要があります．

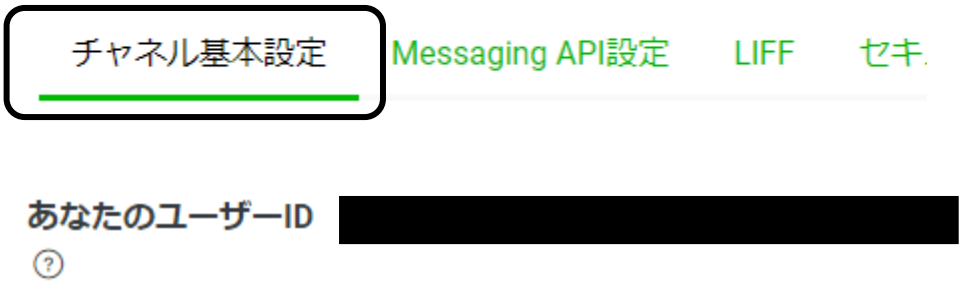


図３．２１　ユーザーＩＤの確認

(3)サンプルプログラムの編集

サンプルプログラム（pushMessage.py）にユーザーＩＤをコピー&ペーストします．

```
#LINEbotAPI インスタンス生成
line_bot_api = LineBotApi(channel_access_token)

def main():
    #ユーザーID をコピーしてください
    user_id = "****************************"

    #プッシュ通知
    line_bot_api.push_message(
        user_id,
        messages=TextSendMessage(text="こんにちは！")
    )

if  __mane__==’__main__’:
    main()
```

(4)動作例

サンプルプログラム（pushMessage.py）を実行し，チャネルにメッセージが通知されていることを確認してください.

図３．２２　メッセージ通知例

３．８　練習問題（通知機能を利用した緊急停止の通知）

(1)通知機能の処理手順

PLC に実装されている MC プロトコルは基本的に PLC 側がサーバであるため，クライアントであるラズベリーパイからコマンドによる問い合わせをすることで稼働状況を取得できます．従って LINEbot にプッシュ通知機能を実装する際にはポーリングにように一定時間ごとに稼働状況の取得処理をする必要があります.

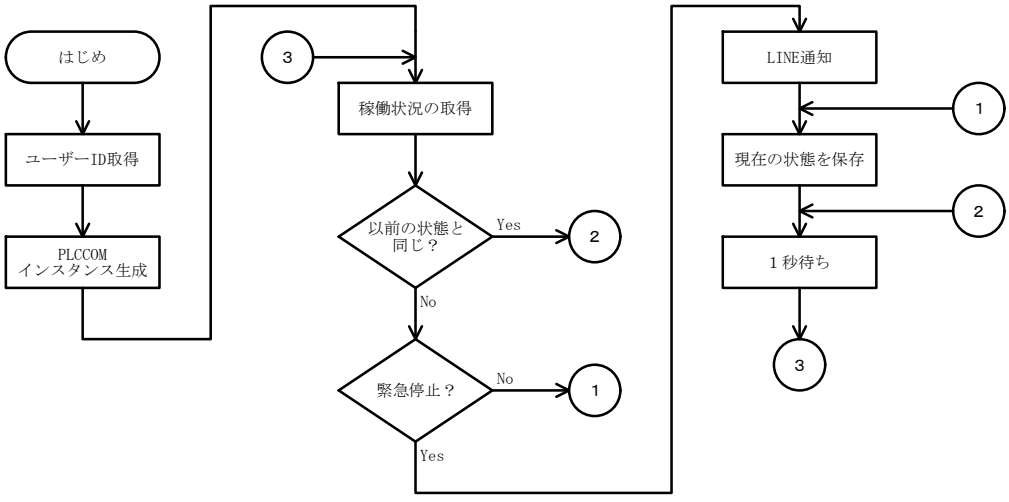


図２．２３　設備の緊急停止による通知アプリのフローチャート

(2)ファイル名，ファイルパス

ファイル名：EmergencyNotification.py

ファイルパス：/home/pi/work/LINE

(3)ヒント

```
…
#設備の保存用変数
old_state = “STOP”

while True:
   #稼働状況の取得
   new_state = ????????????????()

   if ????????????????: #もし old_state と new_state が等しくなければ…
       if ????????????: #もし new_state が”EMERGENCY”だったら…
           #LINE 通知
           ……

       old_state = new_state

   time_sleep(1.0)
```

(4)動作確認

緊急停止アプリを起動させ，設備の緊急停止ボタンを押したときに LINE に通知がされるか確認しましょう．

```
$ python3 EmergencyNotification.py
```

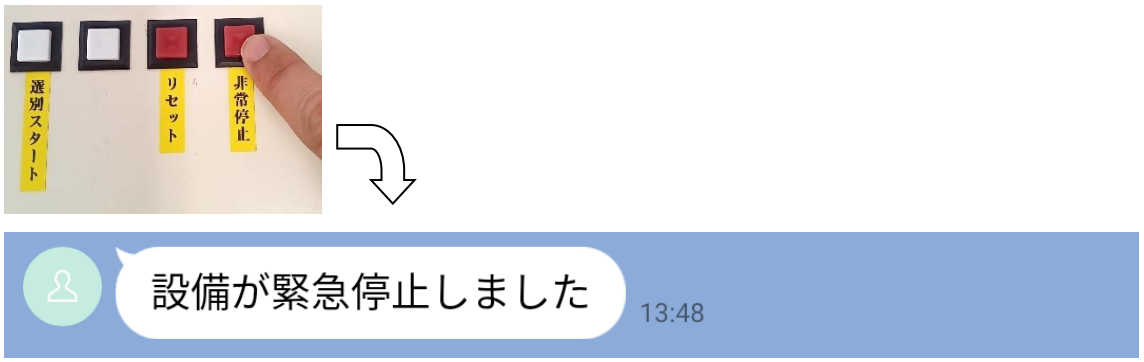


図３．２４　非常停止ボタンを押したことによる通知

(5)追加仕様

・緊急停止の通知以外に，設備が停止した日付と時刻も同時に通知してください.

<ヒント>

Python 言語でシステムの日付・時刻を取得するには datetime ライブラリをインポートします.

```
import datetime

……

#現在の日付・時刻の取得
#(〇〇〇〇/〇〇/〇〇　〇〇:〇〇)形式(文字列)として取得
days = datetime.datetime.now().strftime(“%Y/%m/%d　%H:%M”)

……
```

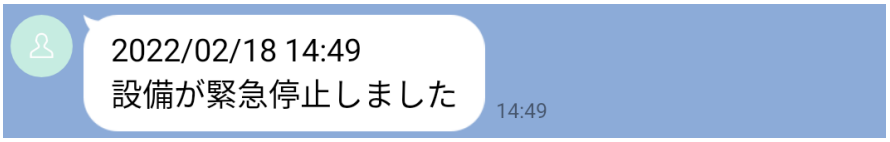


図３．２５　日付・時刻入りの通知

３．９　いろいろな機能の実装
～～ボタンテンプレートによるインターフェースの提供～～

(1)メッセージイベントの弊害

LINEbot ではユーザーからのメッセージを受信することで様々な動作をさせることができます．しかし，ユーザーは送信するメッセージが自由なため，正しいメッセージを送信しないと LINEbot が動作しないという難点があります．とくに英字の大文字と小文字(例えば”stop”と”STOP”)や同一の意味でも異なる表現(「運転状況」と「稼働状況」)があるため，ちょっとした送信間違いなどが発生しやすいといえます．

LINEbot にだれがメッセージを送信しても同一の動作をさせるために LINEbotAPI には**テンプレート**という機能が実装されています．テンプレートには様々なものがありますが今回はボタンテンプレートを紹介します．

(2)テンプレート

テンプレートとは，文字通り LINE プラットフォームにボタンや画像を表示させメッセージ送信を簡易的に行う機能です．つまり文字を１つ１つタップして打ち込む代わりに「表示されるボタンや画像をタップすることであらかじめ決まったメッセージを送信する」という解釈になります．

テンプレートには様々なものがあり，それぞれで使用するモジュールが異なります．例としてボタンテンプレートは下記のモジュールをインポートすることで使用できます．

```
from  linebot.models import (
     ……
      TemplateSendMessage, ButtonsTemplate, MessageTemplateAction,
     ……
)
```

(3)TemplateSendMessage メソッド

あらかじめ決まったメッセージを送信するために TemplateSendMessage メソッドを利用します．TemplateSendMessage メソッドには２つの引数があります．

```
message = TemplateSendMessage (
    alt_text = “****”,
    template = ……
)
```

alt_text ： 代替テキスト．テンプレートメッセージに非対応のデバイスで表示されるほか、ユーザーがメッセージを受信した際に、端末の通知やトークリストでも表示されます．
template：様々なテンプレート（今回はボタンテンプレートを使用）．ボタンテンプレー

トを使用するには TemplateSendMessage メソッドの第二引数である template に ButtonsTemplate メソッドを使用します.

(4) ButtonsTemplate メソッド

表示されるボタンは下記のようなイメージです.

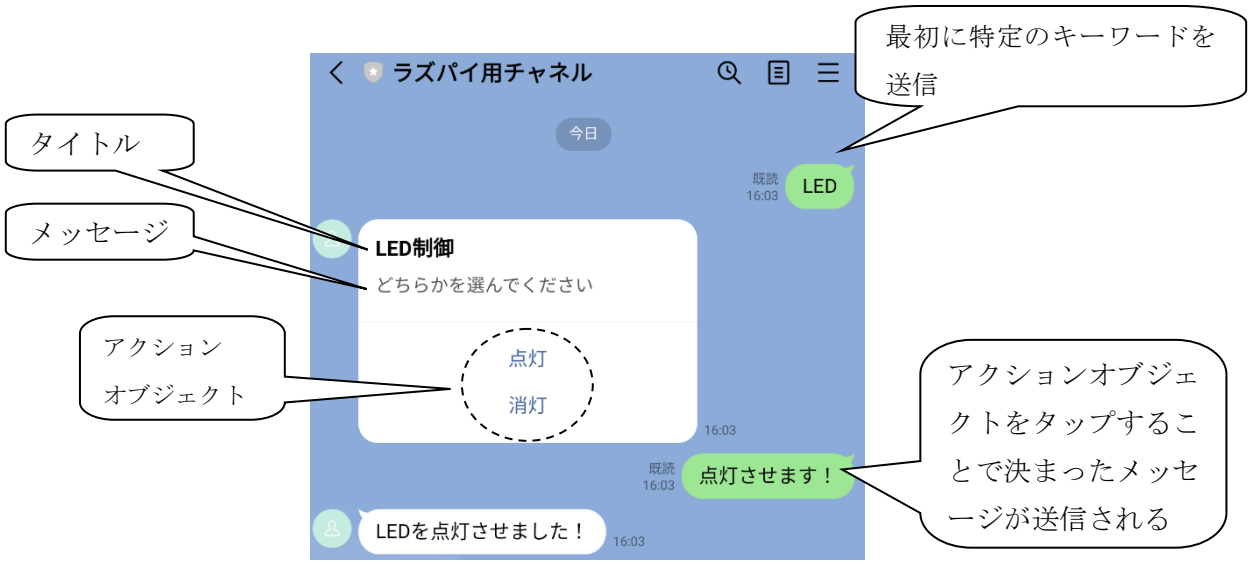


図３．２６　ボタンテンプレートによる表示

上記のようなイメージのボタンを表示させるための ButtonsTemplate メソッドは次の引数を指定します.

```
message = TemplateSendMessage (
    alt_text = “LED”,
    template = ButtonsTemplate (
        text = “どちらかを選んでください”,
        title= “LED 制御”,
        actions = [
            ……
        ]
    )
)
```

メッセージ：選択肢の意味などを説明するために用います

タイトル：端的でわかりやすいタイトルにします

アクションオブジェクト：メッセージや画像がタップした時のアクションを指定する

(5)メッセージアクション

アクションオブジェクトはリスト型で指定します．リスト内のアイテムが増えるほどタップできる選択肢（つまりボタンや画像）が増えると考えてください.

図３．２６のように特定のワードを表示し，その文字をタップすることで決まったメッ

セージを表示させるアクションのことを**メッセージアクション**といいます．メッセージアクションを使用するには MessageTemplateAction メソッドを使用します．

```
message = TemplateSendMessage (
    alt_text = “LED”,
    template = ButtonsTemplate (
        text = “どちらかを選んでください”,
        title= “LED 制御”,
        actions = [
            MessageTemplateAction (
                label=”点灯”,
                text=”点灯させます！”
            ),

            MessageTemplateAction (
                label=”消灯”,
                text=”消灯させます！”
            )
        ]
    )
)
```

タップできる表示ラベル

表示ラベルをタップすると代わりに送信されるメッセージ

MessageTemplateAction メソッド：
１つにつきメッセージ１つ

注意！：アクションオブジェクトは actions 引数にリスト形式で指定します．ただしリストのアイテム数は最大で 4 件までになります！（つまり 1 度に表示できるボタンは最大４つまで）

(6)テンプレートの送信

本来，LINEbot におけるメッセージは JSON 形式で送受信されます(それゆえに Python 以外のプログラム言語でも LINEbotAPI が存在します)．TemplateSendMessage メソッドによって JSON 形式に変更されますので，戻り値が代入された変数を送信することでテンプレートを送信できます．

```
#テンプレートセット
message = TemplateSendMessage (
    ……
)

#テンプレート送信
line_bot_api.relpy_message(
    event.reply_token,
    message
)
```

(6)生産設備管理の適用事例

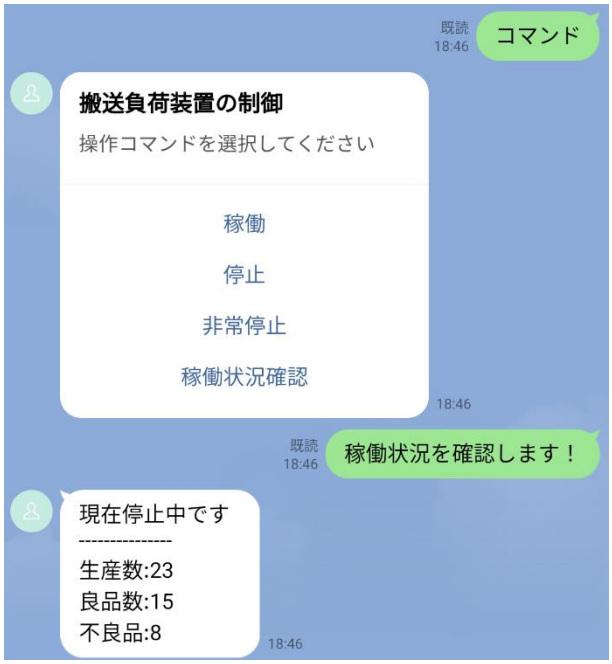


図３．２７　LINEbot を利用した生産設備管理アプリの例

――メモ――

――メモ――