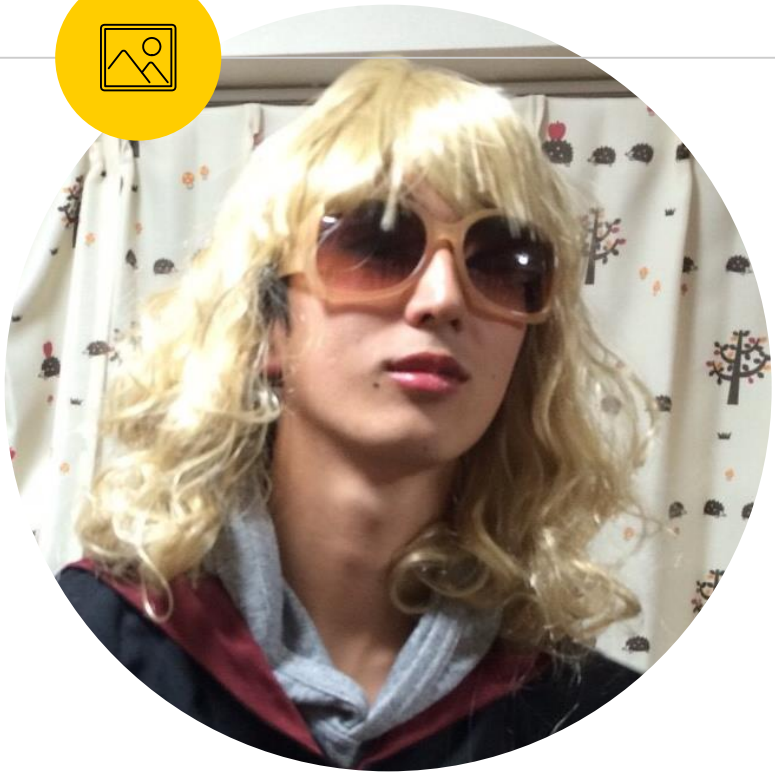
LINE bot
つくってみた

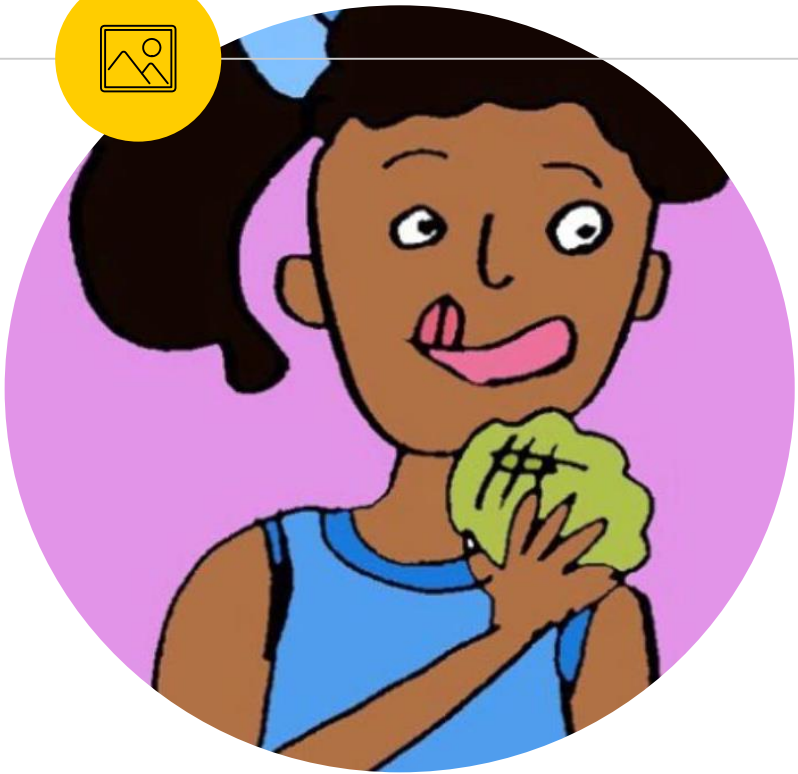


塚本 恭輔　Kyosuke Tsukamoto

所属：テクノロジー本部 デザイン1部 2G

出身：愛知県

趣味：カメラ、筋トレ、テニス、ボルダリング etc.



中野 杏梨　Anri Nakano

所属：テクノロジー本部 メディア開発統括部 Web1G

出身：埼玉県

趣味：イラスト、ひとり旅、作曲、服づくり 読書etc.

目次

- なぜLINE bot?
- LINE botの紹介
- 開発に使った技術の紹介
- ハンズオン！
- 最後に

なぜLINE bot？

研修中の目標

サービスの開発から

リリースまでの流れを理解する！

特にサーバーサイドのことを知りたい！

→ 自分たちで何かサービスを作って学ぼう！

身近なサービスの開発をしたら

開発からリリースまでの流れを実感できるのではないか？

自分たちに身近な**LINE bot**を作ろう！

LINE bot 紹介

女優bot

自分のタイプの女優を紹介

Github URL: https://github.com/tsukamotok/handsOn.git

女優好きのお兄さん

友だち数 17

女優bot　QRコード

開発に使った技術

Node.js

サーバーサイドでjsを動かすための実行環境

特徴

- 非同期処理による高速な動作
- シングルスレッドによるメモリ消費の効率化
- 「V8」エンジンによる高速な実行環境

参考：https://www.sejuku.net/blog/45745

LINE Developers

LINE Botの制作や、既存システムと
LINEを連携したりすることができる
開発者向けのツール

Heroku

PaaS（Platform as a Service）

アプリケーションを実行するための
プラットフォーム

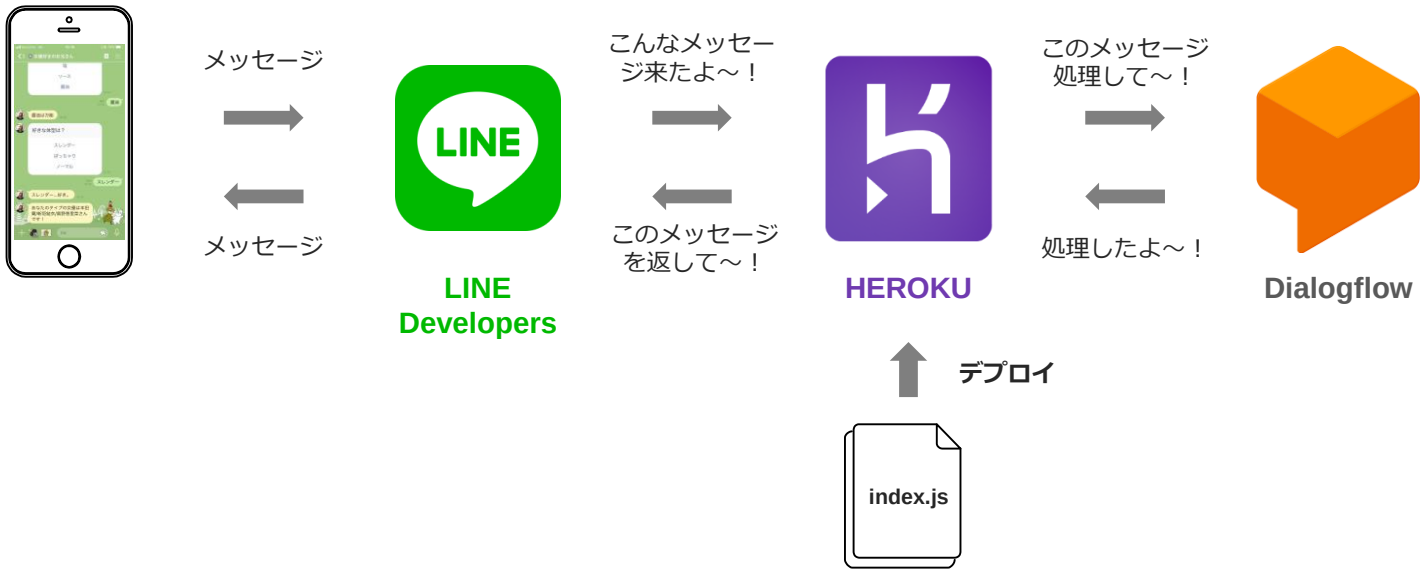
Heroku Git, サーバー

Dialogflow

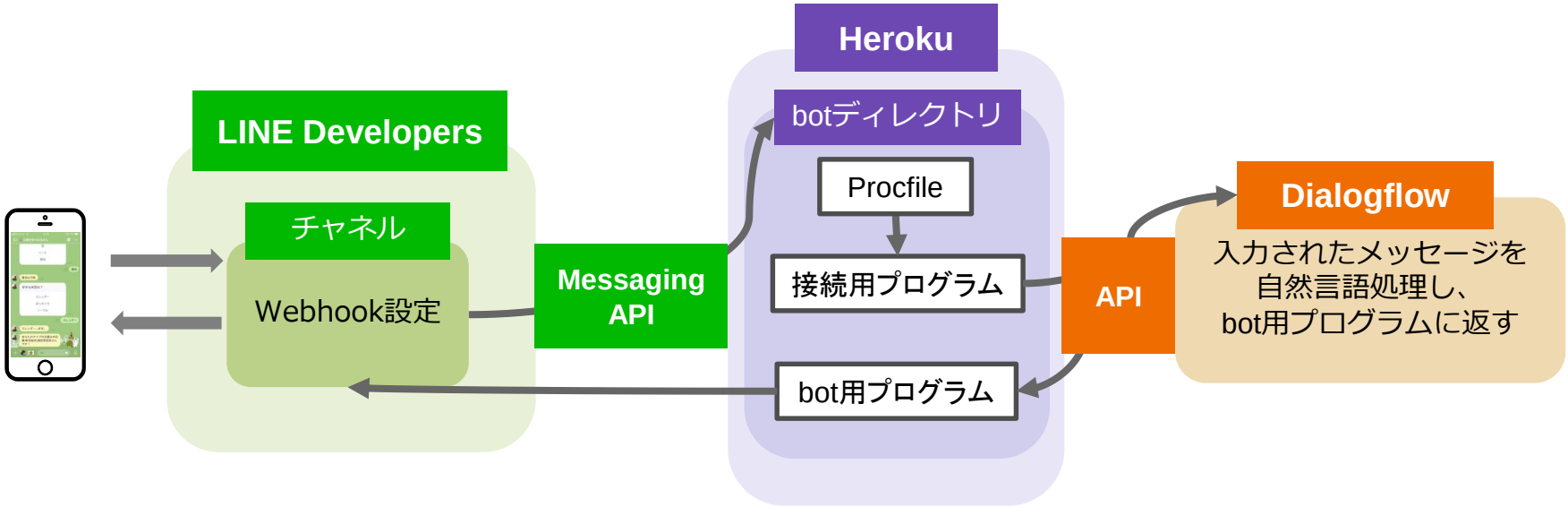
自然言語解析サービス

Botが受け取ったメッセージからユーザーの意図を判定できるようにする。

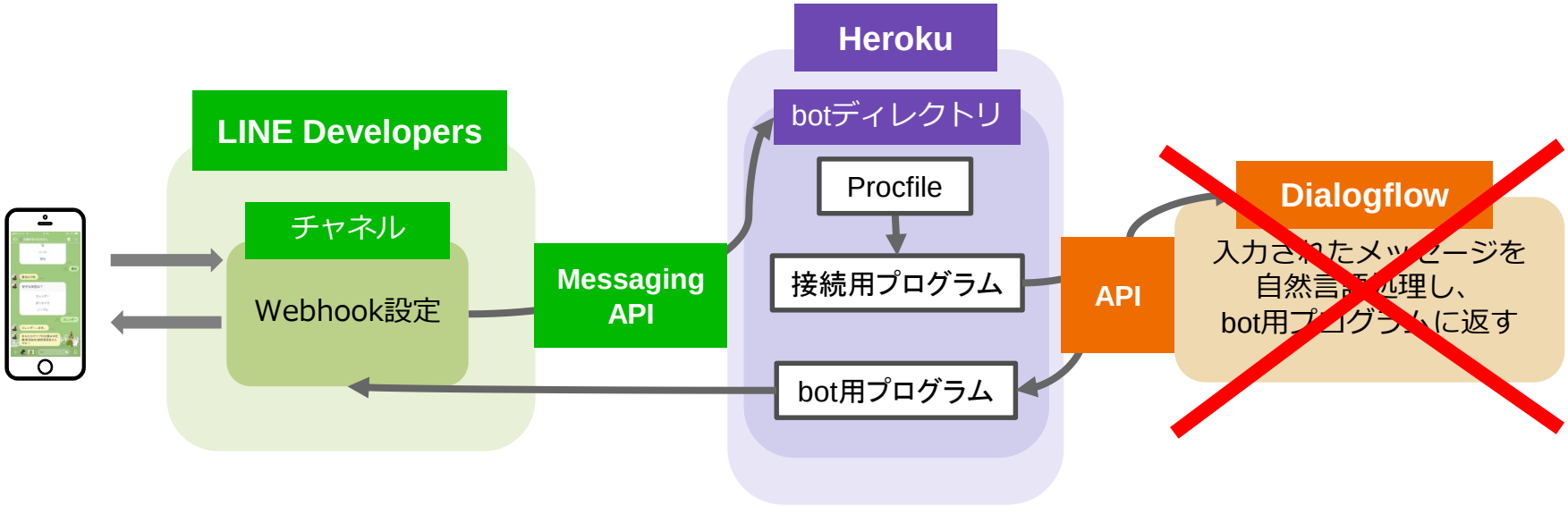
システムの流れ


メッセージ
メッセージ
LINE
LINE
Developers
こんなメッセージ来たよ～！
このメッセージを返して～！
HEROKU
このメッセージ処理して～！
処理したよ～！
Dialogflow
デプロイ
index.js

具体的なシステムの流れ



具体的なシステムの流れ



ハンズオン！

簡単なレスポンスをするbotの作成

開発環境確認

- ✓ **Heroku**アカウント作成済
- ✓ **Heroku CLI**(Command Line Interface)がインストール済
- ✓ **Node.js**がインストール済
- ✓ **LINEプロバイダー**まで登録済

まだの方はこちら

【Windows10版】

Herokuアカウント登録～Heroku CLIインストール
https://note.mu/on_bass/n/n0495484a2b2b

Node.jsインストール方法
https://qiita.com/Masayuki-M/items/840a997a824e18f576d8

LINE開発者/プロバイダー登録画面
https://developers.line.biz/ja/

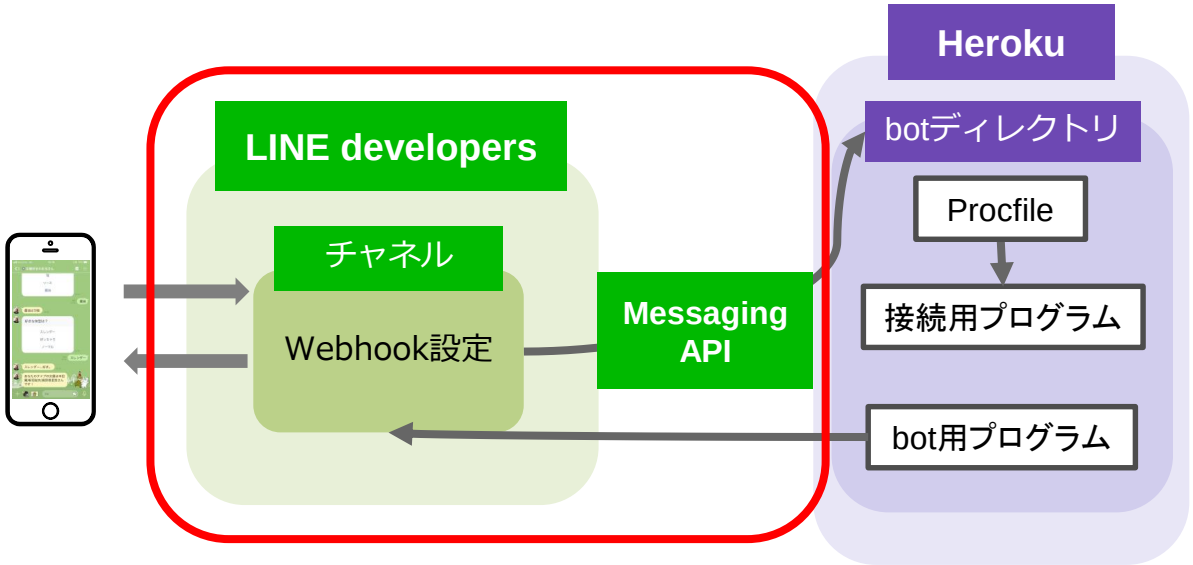
【Mac版】

Herokuアカウント～Heroku CLIインストール
https://blog.katsubemakito.net/macos/setup_heroku-cli

Node.jsインストール方法
https://qiita.com/kyosuke5_20/items/c5f68fc9d89b84c0df09

LINE開発者/プロバイダー登録画面
https://developers.line.biz/ja/

LINE developersでチャネル登録～Messaging API設定



LINE Developers

開発者登録

LINEアカウントで**LINE Developers**にログイン
▼
開発者登録（個人の名前）

プロバイダー

「サービス提供者（企業/個人）」のこと

チャネル
(botアカウント)

LINEプラットフォームが提供する機能を、プロバイダーが開発するサービスで利用するための通信路

Messaging APIを選択

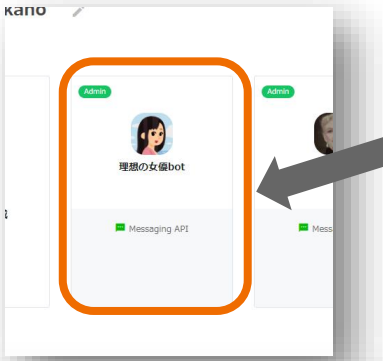
LINEdevelopers https://developers.line.biz/ja/

Channel作成にあたり必要な情報の入力

【Botを作るにあたり必要な情報】
アプリ名: 任意のアプリ名
アプリ説明: 任意のアプリ説明
大業種:任意
小業種: 任意
メールアドレス: ご自身のメールアドレス

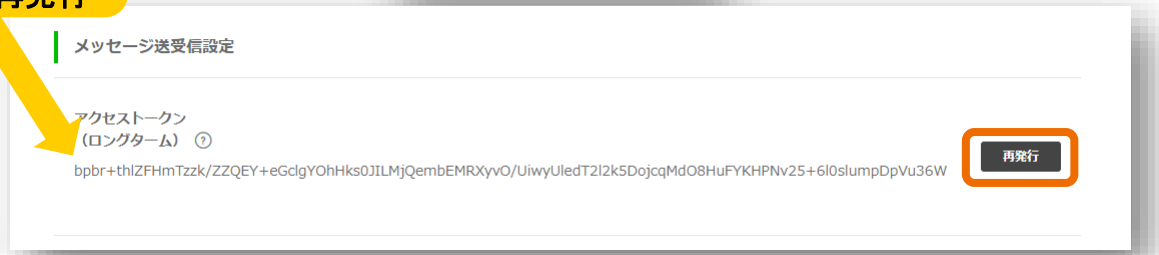
完了まで進んでください！

Channel上の詳細設定①アクセストークンの発行



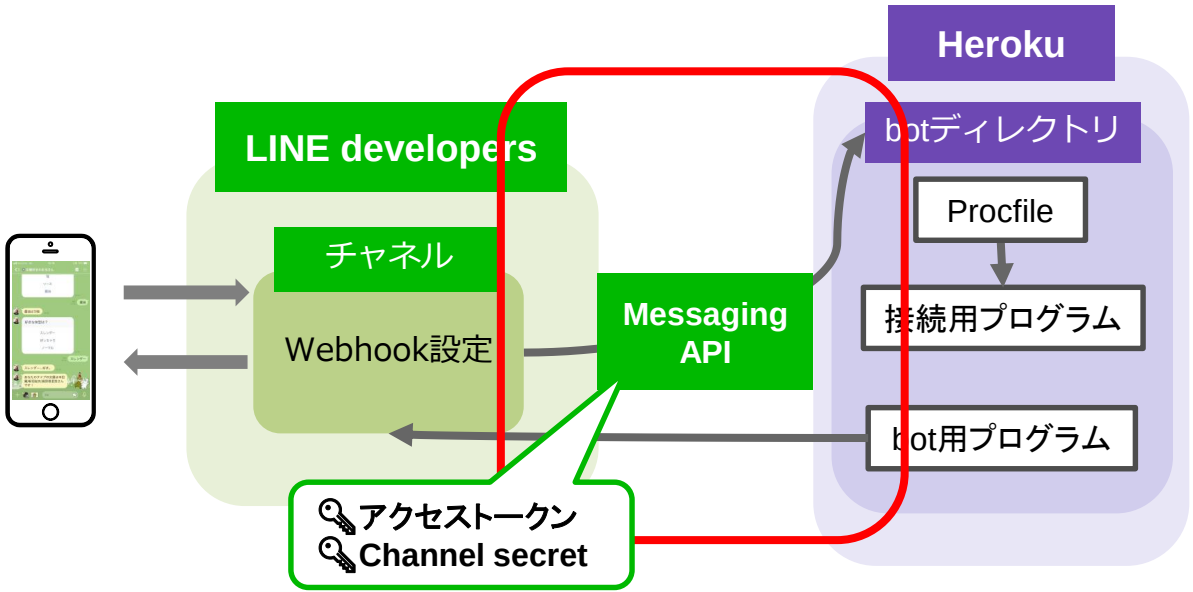
先ほど作った
自分のチャンネルを選択

注意
「Channel ID」ではなく、
「アクセストークン」を再発行



※失効までの時間は**0時間後**で設定

アクセストークンの発行



Channel上の詳細設定　②LINE@機能の利用設定

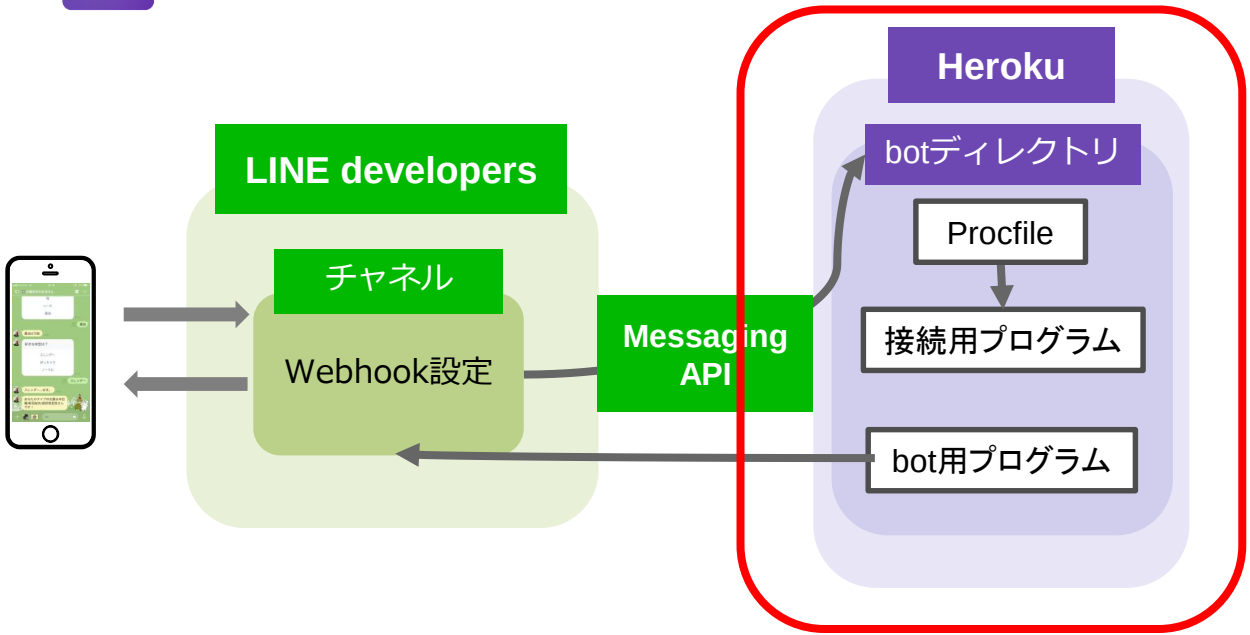
- あいさつメッセージ: オフ
- 応答メッセージ: オフ

Channel上の詳細設定　②LINE@機能の利用設定

※すぐに反映されないことがあります

設定が終了したら設定画面に戻り、ページ更新をして設定が反映されているか確認する

Herokuの設定


Heroku
botディレクトリ
Procfile
接続用プログラム
bot用プログラム
LINE developers
チャネル
Webhook設定
Messaging API

LINEbot本体の開発（Node.jsのプログラム）

```
$mkdir sample-bot   //ソースコードを格納するディレクトリを作成
$cd sample-bot/
```

```
$npm init –yes   //package.jsonファイルを作成
```

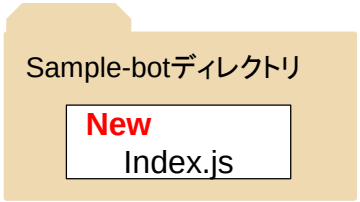
```
$npm install –save express @line/bot-sdk//expressと@line/bot-sdkインストール
```

LINEbot本体の開発（Node.jsのプログラム）

Botの本体とルーター等の設定ファイル

Sample-botディレクトリ内に**index.js**ファイルを新規作成
（お好きなエディタで！）

index.jsファイルにソースコードをコピペ
➡ **【パス】Y:¥個人用¥nakanoan**



サーバーでJavaScript実行するための環境

Index.jsの中身解説(node のプログラム開発)

Index.jsの中身

```
// モジュールのインポート
const server = require("express")();
const line = require("@line/bot-sdk"); // Messaging APIのSDKをインポート

// ----------------------------------------------------------------------
// パラメータ設定
const line_config = {
    channelAccessToken: process.env.LINE_ACCESS_TOKEN, // 環境変数からアクセストークン
    channelSecret: process.env.LINE_CHANNEL_SECRET // 環境変数からChannel Secretをセッ
};

// ----------------------------------------------------------------------
// Webサーバー設定
server.listen(process.env.PORT || 3000);


// ----------------------------------------------------------------------
// ルーター設定
server.post('/bot/webhook', line.middleware(line_config), (req, res, next) => {
    res.sendStatus(200);
    console.log(req.body);
});
```

モジュールのインポート

パラメータ設定
➡環境変数をセットするためのプログラム

Webサーバー設定
➡ポート番号設定

ルータ設定
➡HerokuとMessaging APIを繋げるために必要

Index.jsの中身解説

Index.jsの中身

```
// ------------------------------------------------------------------------------
// ルーター設定
server.post('/bot/webhook', line.middleware(line_config), (req, res, next) => {
    // 先行してLINE側にステータスコード200でレスポンスする。
    res.sendStatus(200);

    // すべてのイベント処理のプロミスを格納する配列。
    let events_processed = [];

    // イベントオブジェクトを順次処理。
    req.body.events.forEach((event) => {
        // この処理の対象をイベントタイプがメッセージで、かつ、テキストタイプだった場合に限定。
        if (event.type == "message" && event.message.type == "text"){
            // ユーザーからのテキストメッセージが「こんにちは」だった場合のみ反応。
            if (event.message.text == "こんにちは"){
                // replyMessage()で返信し、そのプロミスをevents_processedに追加。
                events_processed.push(bot.replyMessage(event.replyToken, {
                    type: "text",
                    text: "これはこれは"
                }));
            }
        }
    });

    // すべてのイベント処理が終了したら何個のイベントが処理されたか出力。
    Promise.all(events_processed).then(
        (response) => {
            console.log(`${response.length} event(s) processed.`);
        }
    );
});
```

ユーザーがテキストメッセージで
「こんにちは」を入力
➡「これはこれは」と返信をする

Bot本体をHerokuにデプロイ準備

```
$git init    //ローカルファイルシステム上でgitリポジトリを初期化する
Initialized empty Git repository in /Users/nkjm/node/sample-bot/.git/
```

デプロイに必要なファイルを作成

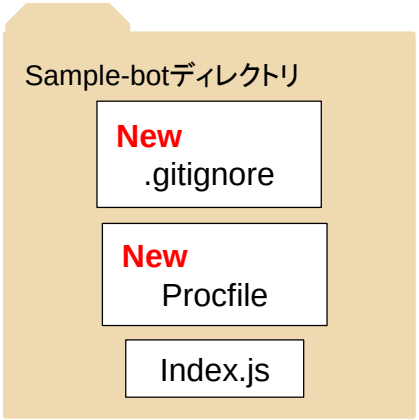
sample-botディレクトリに**.gitignore**と**Procfile**を新規作成

.gitignoreの中身 ※コピペ

```
//gitの管理対象外にしたいファイル
npm-debug.log
node_modules
```

Procfileの中身 ※コピペ

```
//Herokuにプログラムの起動方法を教えるための設定ファイル
web:  node index.js
```

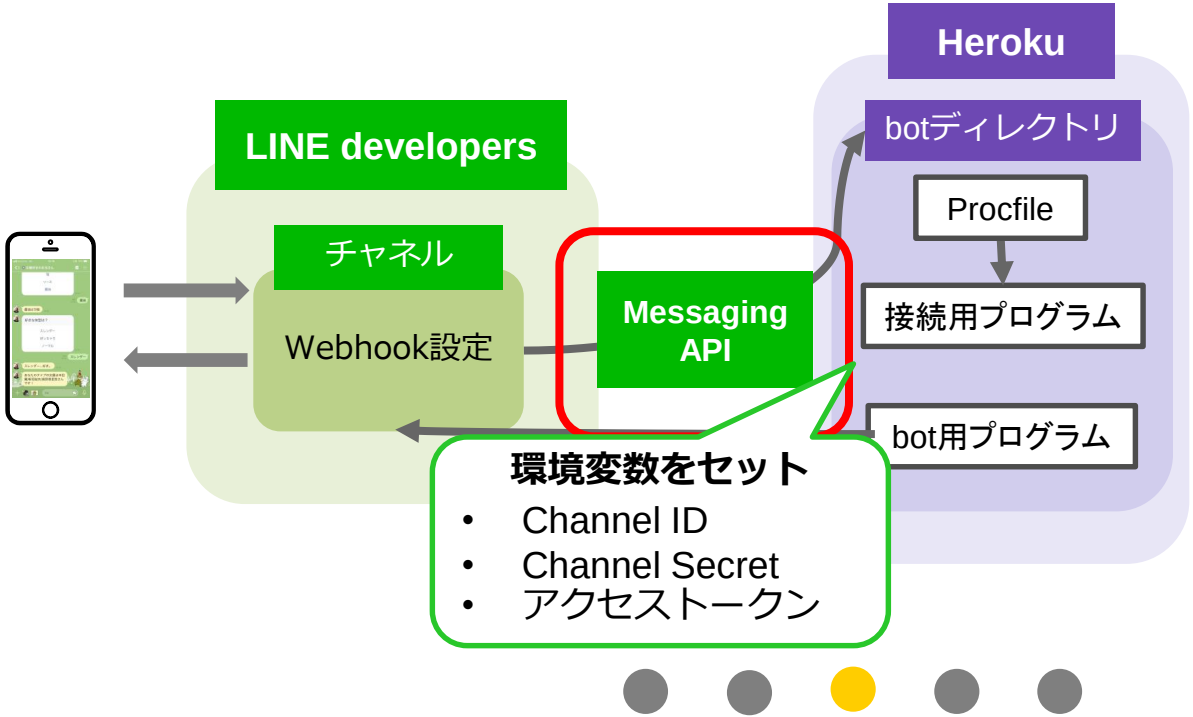


Bot本体をHerokuにデプロイ準備（herokuにログイン&アプリ作成）

$ heroku login//heroku CLIを使ってherokuにログイン
Enter your Heroku credentials.
Email: あなたのEmail
Password (typing will be hidden):
Logged in as あなたのEmail

$ heroku apps:create あなたのアプリ名//Heroku上にアプリケーションを作成
Creating ⬢ あなたのアプリ名... done
https://あなたのアプリ名.herokuapp.com/ | https://git.heroku.com/あなたのアプリ名.git

LINEとHerokuをつなぐ設定

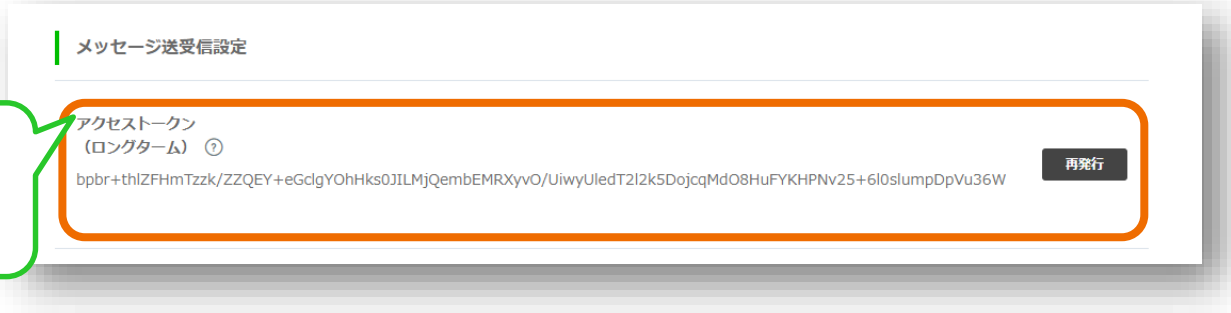


Bot本体をHerokuにデプロイ（準備）

```
//Messaging APIのSDKで必要となる環境変数をセット
$ heroku config:set LINE_CHANNEL_ID=あなたのChannel ID
$ heroku config:set LINE_CHANNEL_SECRET=あなたのChannel Secret
$ heroku config:set LINE_ACCESS_TOKEN=あなたのアクセストークン
```

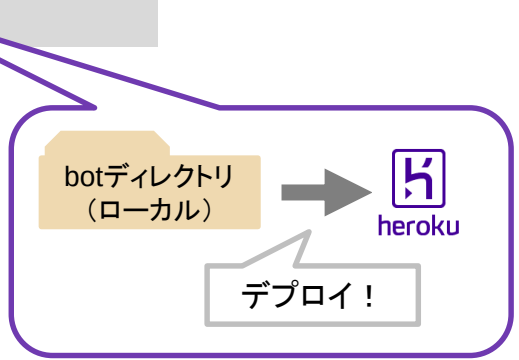
LINE developersの設定画面より

- Channel ID
- Channel Secret
- アクセストークン

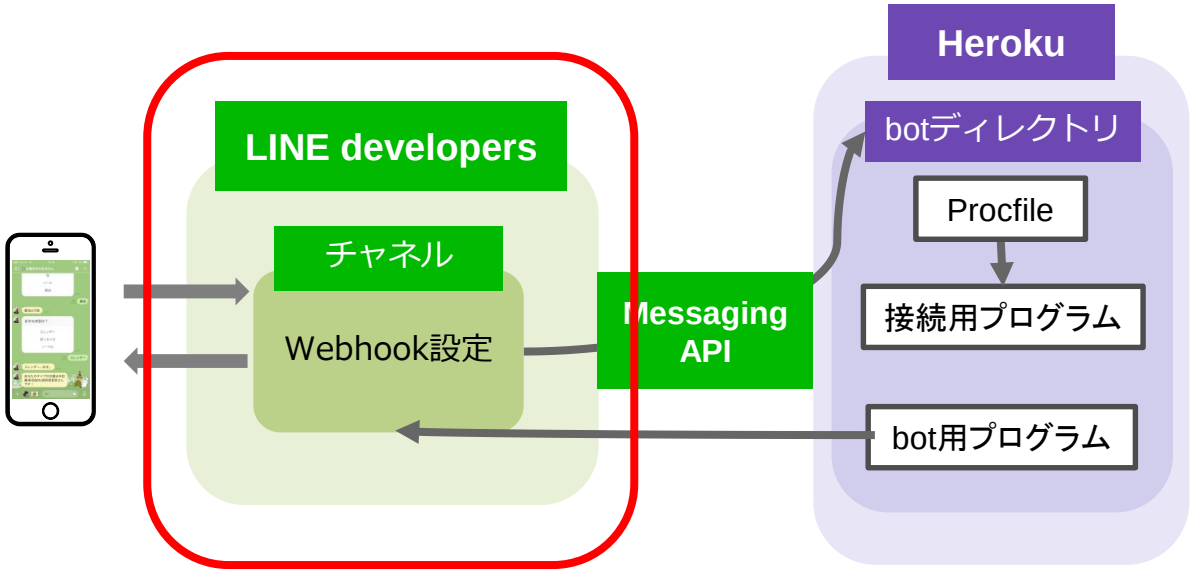


Bot本体をHerokuにデプロイ

```
//これまで作成したコードをレポジトリにコミットし、デプロイ
$ git add .
$ git commit -m "First commit"
$ git push -u heroku master
// 途中出力省略
remote: Verifying deploy.... done.
To https://git.heroku.com/あなたのアプリ名.git
 * [new branch]      master -> master
```



Webhookを設定


Heroku
botディレクトリ
Procfile
接続用プログラム
bot用プログラム
LINE developers
チャネル
Webhook設定
Messaging API

Webhookとは？

- WebhookはLINEのMessaging APIを構成する要素の１つ
- ユーザから送信されたメッセージをリアルタイムに受信する仕組みのこと
- 使うためにはWebhookのアクセスポイントを設定しておく必要がある
- →　Webhook URL

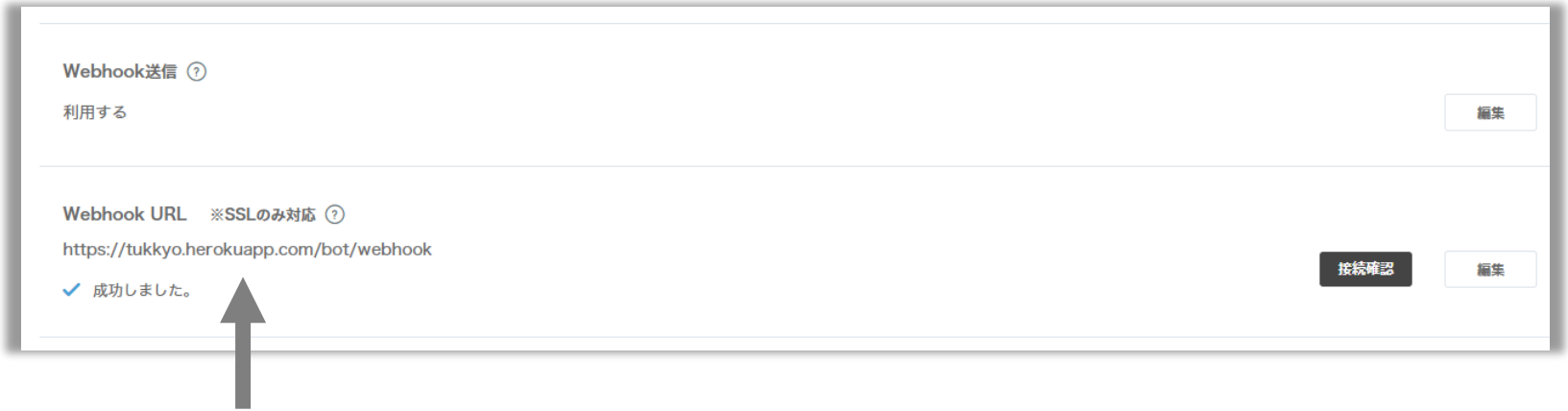
Webhookを設定

自動応答メッセージののの設定はこちらを押し、
詳細設定のWebhookを「オン」にする。

Webhookを設定

Webhook送信の編集を押し、利用するにして更新

Webhookを設定

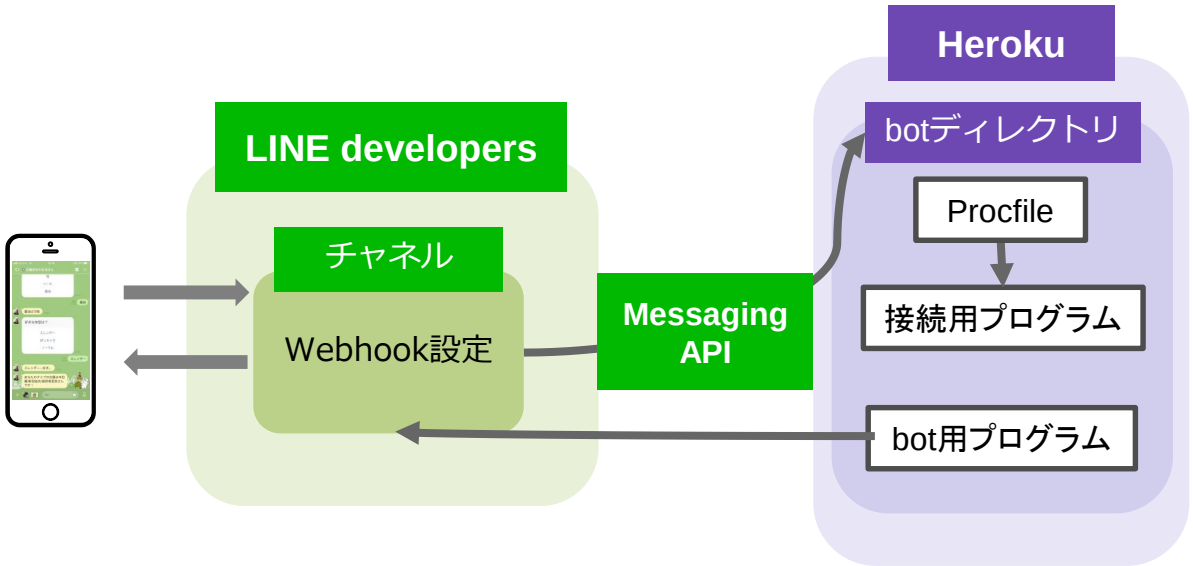


▼Webhook URL

https:// **hetokuで設定したあなたのアプリ名**.herokuapp.com/bot/webhook

接続確認を押し、成功しましたと出たら完了！

Webhookを設定


Heroku
botディレクトリ
Procfile
接続用プログラム
bot用プログラム
LINE developers
チャネル
Webhook設定
Messaging API

作ったLINEbotを友達追加

➡実際にbotを動かしてみてください！

メッセージの確認

このように動くでしょうか？

☞早く終わった方はこちら！

【応用編】index.jsの修正で応答メッセージを変更！

```
req.body.events.forEach((event) => {
    // この処理の対象をイベントタイプがメッセージで、かつ、テキストタイプだった場合に限定。
    if (event.type == "message" && event.message.type == "text"){
        // ユーザーからのテキストメッセージが「こんにちは」だった場合のみ反応。
        if (event.message.text == "こんにちは"){
            // replyMessage()で返信し、そのプロミスをevents_processedに追加。
            events_processed.push(bot.replyMessage(event.
                type: "text",
                text: "これはこれは"
            }));
```

「こんにちは」と「これはこれは」を変更してみてください！

//デプロイコマンド
$ git add .
$ git commit -m “Second commit"
$ git push -u heroku master

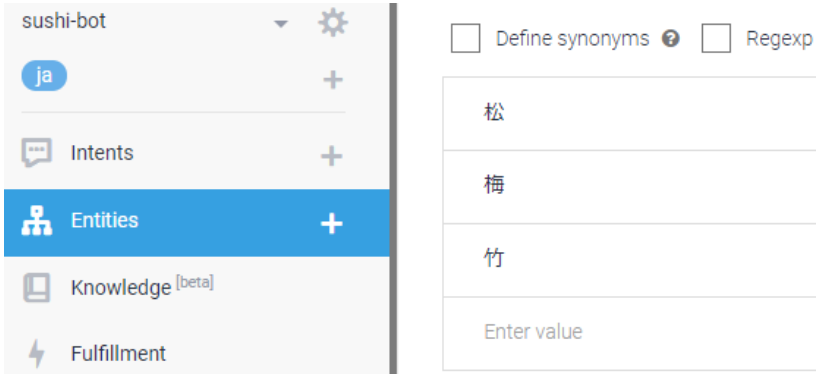
最後に

学んだこと

- サービスの開発からリリースまでの流れ
- コマンドラインの操作（git, npm, Heroku etc.）
- プログラムとソフトウェアをつなぐ際に必要なもの（SDK, Webhook etc.）

Dialogflowのすごいところ

任意のキーワードを抽出してくれる



詰まったところ

Webhookの設定（Qiita）

詰まったところ

Webhookの設定（僕たちのLINE Developers）

詰まったところ

最初から手順をやり直してもうまくいかない…

詰まったところ

Webhookの設定（Qiita）

URLが違う！

Index.js（Qiita）

```
// ルーター設定
server.post('/bot/webhook', line.middleware(line_config), (req, res, next) => {
    // 先行してLINE側にステータスコード200でレスポンスする。
    res.sendStatus(200);
```

詰まったところ

URL変更後、正常に作動

詰まったところ

そのものの仕組みを理解していれば、エラーが起きても

どの部分に問題があるか見つけることができる。

分かっていないと無駄足に…

紹介

LINE Front-end Framework (LIFF)

Massaging APIの新機能

LINE内で動作するウェブアプリのプラットフォーム
LIFFに登録したウェブアプリをLINE内で起動すると、LINEのユーザーIDなどをLINEプラットフォームから取得できる。

こんなLINEbotがあったらいいな　@cosme × LINEbot

入力したキーワードに沿って
動画/メイク方法の記事が送られてくる

参考資料

LINE botの作り方

（前編）https://qiita.com/nkjm/items/38808bbc97d6927837cd

（後編）https://qiita.com/nkjm/items/4de41988969e6f17adcb

Massaging APIについて

https://developers.line.biz/ja/docs/messaging-api/overview/

LIFFについて

https://developers.line.biz/ja/docs/liff/overview/