

LINE Botをつくってみよう

APIを試して学んでしっかりわかる

mochikoAsTech 著

2023-05-20 版

mochikoAsTech 発行

はじめに

2023 年 5 月 mochikoAsTech

この本を手にとってくださったあなた、こんにちは、あるいははじめまして。「LINE Bot をつくってみよう 〜API を試して学んでしっかりわかる〜」の筆者、mochikoAsTech です。

いきなりですが、プログラムを書いたことはありますか？自分の書いたプログラムを動かしてみ、ブラウザで結果が表示されたときの「う、動いたー！」という喜びっていいですね。まだ書いたことがない、という人には「あれはいいものだぞ！」とお勧めしておきます。

真っ白い背景に黒字で **Hello World!** と表示されただけでも「自分が作ったものがいつも使っているブラウザで動いたー！」という喜びでいっぱいになるのに、それが「自分が作ったものがいつも使っている LINE で動いた！」になるとますます嬉しいので、筆者はじめてプログラミングをやってみるなら LINE Bot を題材にするのはなかなかよい選択じゃないかと思っています。無料ではじめられるし、たぶんあなたが思ってるより少ない手順であっさり動きます。それでいてこだわろうと思ったら色んな奥深い機能があって飽きないし、ドキュメントが日本語であるし、コミュニティも活発で質問や相談がしやすいです。

あ、それからいま色んなところで話題になっている ChatGPT も触ってみたいと思いませんか？なんかすごいらしいし、ちょっと気になっている…それなら「気になっている」まま時間が経ってしまうともったいないので、ここでガッと触ってみましょう。他人が触った話を聞いたり読んだりするのもいいですが、やっぱり自分で触てみると情報量も理解度も格段に違います。「やってみようかな」や「やってみたいな」という衝動は、生まれた瞬間にグッと押込んで躊躇わずそのまま飛び込んでみると楽しいですよ！

あなたも本書で「気になっていた ChatGPT を使って AI チャットボットが作れた！自分が作ったものが LINE で動いた！」という喜びを味わってみませんか？大事なのは勢いです。「つつこむぞ、つかまれ！」という気持ちではじまりはじまりー！

想定する読者層

本書は、こんな人に向けて書かれています。

- ・「API をたたく」をやってみたい人
- ・「チャットボットを作る」をやってみたい人
- ・これからプログラミングを学ぼうと思っている人
- ・LINE API を使ってみたい人
- ・ChatGPT や GPT-3.5、GPT-4 が気になっている人
- ・LINE 公式アカウントの運用をしている人
- ・LINE の Messaging API がどういうものか知らない人

マッチしない読者層

本書は、こんな人が読むと恐らく「not for me だった…（私向けじゃなかった）」となります。

- ・コミュニケーションアプリ「LINE（ライン）」を使ったことがなく、LINE の基本操作から知りたい人
- ・LINE 公式アカウントで友だちをうまく増やす広告や宣伝の方法を知りたい人

本書の構成

本書は、第 1 章「LINE 公式アカウントをつくってみよう」で LINE 公式アカウントについて解説し、第 2 章「Messaging API で LINE Bot をつくってみよう」で実際に LINE Bot をつくって動かし、最後に第 3 章「Messaging API の色んな機能を試してみよう」で LINE Bot を育てていく色んな方向性を紹介する、という三部構成になっています。

本書の特徴

実際に LINE 公式アカウントからメッセージを送ったり、オウム返しする LINE Bot や AI チャットボットを作ったりします。手を動かして試してから裏付けとなる事柄を学ぶ、というステップを交互に踏むため、理解がしやすく、プログラミング初心者でも飽きずに読み進められる内容です。

また実際の開発でありがちなトラブルをとり上げて、

- こんなエラーが起きたら原因はどう調べたらいいのか？
- 問題をどう解決したらいいのか？
- どうしたら事前に避けられるのか？

を解説しています。

本書はフルカラーの PDF 版をどなたでも無料でダウンロードできますので、ぜひご活用ください。詳しくは「あとがき」にある「PDF 版のダウンロード」を参照してください。

用語の定義

私たちは「LINE（ライン）」という言葉で、無意識に色んな意味で使っています。

「LINE ってやってる？」「うん。でも家族としか使っていない」というときの LINE は、コミュニケーションアプリとしての LINE アプリのことを指しています。「昨日送った LINE 見た？」「ごめん。寝てた」というときの LINE は、LINE アプリのトーク上でやりとりするメッセージのことを指しています。「LINE のオフィスってカフェがあるらしいよ」「いいねー」というときの LINE は、LINE 株式会社のことを指しています。その他にも LINE Pay や LINE MUSIC といったさまざまな LINE 関連サービスや機能が、略称として LINE と呼ばれているケースもあります。

こんなふうに、同じ「LINE」という言葉でも文脈によって色んな意味になるため、本書における用語を最初に定義しておきます。

- 本書では、LINE アプリのことを「LINE」と呼びます。ただ「LINE」とだけ書いてあったら、これは LINE アプリのことだな、と思ってください。
- LINE アプリのトーク上でやりとりするメッセージのことは、「メッセージ」と呼びます。
- 会社としての LINE は、「LINE 株式会社」と呼びます。
- それ以外の各サービスは「LINE Pay」や「LINE MUSIC」のような正式名称で呼びます。

用意するもの

- LINE がインストールされたスマートフォン
- パソコン（Windows または Mac）

本書のゴール

本書を読み終わると、このような状態になっています。

- LINE 公式アカウントについて一通りの知識がある
- LINE Bot が動く仕組みを理解している
- LINE Bot を作るときに何を準備すればいいか分かっている
- Webhook と API の違いが分かっている
- プロバイダーやチャンネルに何を登録すべきか分かっている
- 読む前より LINE API が好きになっている

免責事項

本書に記載された社名、製品名およびサービス名は、各社の登録商標または商標です。本書において操作説明のために載せている各画面のスクリーンショットは、著作権法第32条引用および過去の判例に従い、出典元を明記した上で、必然性のある箇所でのみ、引用と分かる形で掲載しています。

本書の発刊時点で筆者は LINE 株式会社に所属していますが、本書は個人として執筆したものであり、本書に記載されている内容はいずれも所属する組織の公式見解ではありません。また、本書は一般に開示されている情報を元に書かれており、筆者が所属する組織において知り得た情報は含まれていません。LINE API について言及するにあたって、所属を隠した宣伝であると誤解されないよう所属組織をここに明記しておきます。

本書はできるだけ正確を期すように努めましたが、筆者が内容を保証するものではありません。よって本書の記載内容に基づいて読者が行なった行為、及び読者が被った損害について筆者は何ら責任を負うものではありません。

不正確あるいは誤認と思われる箇所がありましたら、必要に応じて適宜改訂を行いますので GitHub の Issue や Pull request で筆者までお知らせいただけますと幸いです。

<https://github.com/mochikoAsTech/startLINEBot>

CONTENTS

CHAPTER 1

LINE公式アカウントをつくってみよう

- 1.1 LINE Botをつくる前に14
- 1.2 LINE公式アカウントとは15
 - 1.2.1 LINE公式アカウントにかかる費用16
- 1.3 LINE公式アカウントを作る17
 - 1.3.1 LINEでLINE公式アカウントを作ってみる18
- 1.4 LINE公式アカウントからメッセージを送る25
 - 1.4.1 管理アプリを使ってLINE公式アカウントからメッセージを送ってみる26
 - コラム** 個人のLINEアカウントをLINE公式アカウントに切り替えられる?33
 - 1.4.2 LINE公式アカウントと友だちになる34
 - コラム** 開発用途のLINE公式アカウントが知らない人に友だち追加されてしまった。ブロックできる?37
 - 1.4.3 LINE公式アカウントから友だちにメッセージを送ってみる38
 - コラム** LINE公式アカウントのメッセージはさかのぼって見られる?42
 - コラム** メッセージの通数はどうカウントされるのか43
 - 1.4.4 LINE公式アカウントにメッセージを送ってみる44
- 1.5 LINE公式アカウントの基礎知識46
 - 1.5.1 LINE公式アカウントと普通のLINEアカウントとの違い46
 - 1.5.2 プレミアムIDとベーシックID48
 - 1.5.3 未認証アカウントと認証済アカウント49
 - コラム** 準備中のLINE公式アカウントをリリース日まで非公開にしておける?50

Messaging APIでLINE Botをつくってみよう

2.1 LINE公式アカウントをチャットボットにしたい52

2.1.1 チャットボットとは52

2.2 Messaging APIでチャットボットを作るには52

2.2.1 APIとは53

コラム APIは「たたく」もの? 「呼び出す」もの?55

2.2.2 Messaging APIとは55

2.3 Messaging APIチャンネルを作ろう56

2.3.1 LINE Official Account Managerにログインする57

2.3.2 Messaging APIチャンネルを作って紐づける60

2.4 Messaging APIを使ったメッセージ送信を試してみよう67

2.4.1 LINE Developersコンソールでチャンネルアクセストークンを発行する67

2.4.2 Messaging APIでブロードキャストメッセージを送信する71

コラム 仕事でLINE Botを開発するときにも「個人のLINEアカウント」が必要か?78

2.5 LINE公式アカウントから友だちに返信するには78

2.5.1 方法1. 固定の自動応答を設定しておいて個別対応は一切しない79

2.5.2 方法2. 人間が手打ちのチャットで返信する80

2.5.3 方法3. 内容に応じた自動応答メッセージで応対する80

2.5.4 方法4. Messaging APIで返信する81

2.5.5 Webhookとは81

コラム LINE公式アカウントの「既読」はいつ付くのか?83

2.6	オウム返しするチャットボットを作ってみよう	83
2.6.1	Messaging APIのSDKを準備する	84
2.6.2	AWS LambdaとAPI Gatewayでボットサーバーを作る	94
2.6.3	Webhook URLを設定する	111
2.6.4	友だち追加されたときのあいさつメッセージを併用する	115
2.6.5	LINEプラットフォームから飛んできたWebhookを目視確認する	115
2.6.6	LINE公式アカウントに話しかけてオウム返しを確認しよう	118
コラム	これは本当にLINEプラットフォームから来たWebhook？	118
2.7	OpenAIのAPIを使ったAIチャットボットを作ってみよう	120
2.7.1	ChatGPTとGPT-3.5とは	120
2.7.2	OpenAIに登録してシークレットキーを取得する	121
2.7.3	OpenAI APIに質問を投げて回答を取得する	127
2.7.4	OpenAI APIのSDKを準備する	130
2.7.5	OpenAI API SDKのレイヤーを作成する	138
2.7.6	AWS Lambdaのコードに 「OpenAIのAPIで質問の回答を取得する処理」を追加する	144
2.7.7	Lambda関数のタイムアウトまでの時間を延ばす	148
2.7.8	環境変数にOpenAIのシークレットキーを追加する	150
2.7.9	LINE公式アカウントに話しかけてAIチャットボットの回答を確認しよう	152
コラム	Webhookへのレスポンスが先？ 応答メッセージが先？	153

Messaging APIの色々な機能を試してみよう

3.1 メッセージ送信に関する機能156

3.1.1 ユーザーIDを指定して特定の人に送る156

コラム 開発チームだけにメッセージのテスト配信がしたい156

3.1.2 メッセージの配信対象を属性で絞り込む157

3.1.3 メッセージ送信元のアイコンと表示名を変更する158

コラム URLを送る前にOGPの見た目を確認したり、キャッシュを消したりしたい160

3.1.4 Flex Messageで見た目にこだわったメッセージを送る161

3.1.5 ボットサーバーがWebhookを受け取れなかったときの再送機能163

3.2 リッチメニュー164

3.2.1 リッチメニュープレイグラウンドでリッチメニューを体験してみよう165

3.2.2 リッチメニューの画像を作る167

3.2.3 LINE Official Account ManagerとMessaging APIでの
リッチメニューの制約の違い168

3.3 Messaging APIをもっと楽しむために168

3.3.1 Messaging API以外のプロダクトとの組み合わせ168

3.3.2 LINEキャンパスで資格を取ってみよう169

3.3.3 LINE Creative Labを使おう169

3.3.4 パソコン版のLINEを使おう169

3.3.5 LINE Developers Communityに参加しよう169

あとがき171

PDF版のダウンロード172

著者紹介175

CHAPTER 1

LINE公式アカウントを つくってみよう

あなたにも、私にも。

LINE公式アカウントは誰にでもつくれる！

1.1 LINE Botをつくる前に

「LINE Bot をつくってみよう」という本なのに、本編を読みはじめたらいきなり第1章で「LINE 公式アカウントをつくってみよう」と書いてあると戸惑いますよね。

実は、話しかけて自動応答する LINE Bot を作るには、話しかける相手として「LINE 公式アカウント」というものを用意する必要があります。「いま使っている LINE のアカウントをそのまま LINE Bot にできないの？」と思われるかもしれませんが、個人の LINE アカウントを操作するための API^{*1}は提供されていないため、必ず「LINE 公式アカウント」が必要なのです。そのため第1章では、この LINE 公式アカウントの作り方や、LINE 公式アカウントに関する最低限の知識を説明していきます。

でも晩ごはんは肉じゃがにしようと思い立って「肉じゃがをつくろう」という本を開いたのに、前半が「じゃがいもの選び方」「スーパーマーケットと八百屋のどっちで買うべき?」「肉じゃがに適した雪平鍋を用意しよう」などで埋まっていてなかなかレシピが出てこない、「そういうのいいから! 早く! 肉じゃがのレシピ!!」と感じると思います。お気持ちは分かります。ただ自動応答する LINE Bot をつくるにあたって、先に LINE 公式アカウントを用意すると共に、LINE 公式アカウントの「中の人」として手動の運用をざっくり体験しておいてもらうことで、自動応答の LINE Bot をつくったときに「あー、さっき手でやったあの作業をコードのこの部分でやらせてるんだ!」が分かりやすくなるので、敢えてこういった構成にしています。

第1章を読んでから第2章、という順番で読み進めてもらうのがお勧めではありますが、「細かいことはいいから! 早く LINE Bot を作らせて!」という方は、「1.3.1 LINE で LINE 公式アカウントを作ってみる」だけやったら、あとは読み飛ばして第2章「Messaging API で LINE Bot をつくってみよう」に進んでも構いません。

それでは今度こそ、はじまりははじまりー!^{*2}

^{*1} API については、「2.2.1 API とは」で後述します。

^{*2} 本書は LINE 公式アカウントや LINE Bot や Messaging API について、筆者が書きたいこと伝えたいことをぎゅうぎゅうに詰め込んで作った本です。「なんでこの話、いまここでするんだろう?」と感じたら、そのときは「たぶん…ただ書きたかったんだな…」と思って、情報量の調整にしくじったヲタクの早口を聞くつもりで温かく見守ってください。よろしくお願いします。

1.2 LINE 公式アカウントとは

服屋さんのレジで「LINEで友だちになると5%オフになるクーポンがもらえるんですが、よかったらどうですか？」と声をかけられたり、カフェでテーブルの上に「友だち追加するとワンドリンク無料！」と書かれたポップが置いてあったりするのを見たことはありませんか？（図 1.1）



▲図 1.1 街中で見かける LINE 公式アカウント

普段、あなたは LINE で友だちとメッセージや写真をやりとりしていると思います。実在する人間の友だちとやりとりするのと同じように、LINE では企業や店舗ともやりとりできます。この企業や店舗のアカウントが「LINE 公式アカウント」*3と呼ばれるものです。

LINE 公式アカウントには、企業や店舗だけでなく、芸能人のものも存在します。たとえば地震のときに誰よりも早く「大丈夫？」とメッセージをくれて、彼氏感がすごい、と

*3 以前の「LINE@（ラインアット）」という名称の方が覚えがある、という人も居るかもしれません。LINE@は 2019 年 4 月 18 日に LINE 公式アカウントへ統合されたのですが、いまだに LINE@の名前で友だちを募集している店頭ポスターを見ることも少なくありません。 <https://d.line-scdn.net/stf/line-lp/migrationinformation-LINEAT190718.pdf>

話題になった芸能人の LINE 公式アカウント^{*4}もありました。

この LINE 公式アカウントは、企業や店舗などの法人、あるいは芸能人のような「特別に許可を得た人」だけでなく、誰でも、つまりあなたでも無料ですぐに作ることができます。

1.2.1 LINE 公式アカウントにかかる費用

無料ですぐに作れると言われても、本当にお金がかからないのか気になりますよね。LINE 公式アカウントにかかる費用について説明しておきましょう。

LINE 公式アカウントには、2023 年 6 月時点で 3 つのプランがあります。それぞれのプランで使える機能に差はなく、無料メッセージの通数や、無料メッセージを使い切ったときに、さらに追加でメッセージが送れるか、という点でのみ違いがあります。(図 1.2)

	料金	メッセージ通数
コミュニケーションプラン	0円	200通/月
ライトプラン	月額 5,000円	5,000通/月
スタンダードプラン	月額 15,000円	30,000通/月 [追加 1通3円]

▲図 1.2 2023 年 6 月時点の LINE 公式アカウントの料金プラン

2023 年 5 月 31 日までは、フリープラン、ライトプラン、スタンダードプランの 3 種類でしたが、2023 年 6 月 1 日^{*5}からフリープランの名称がコミュニケーションプランに代

^{*4} これは好きになっちゃう…。地震直後に送られた佐藤健の LINE が話題に | BuzzFeed - バズフィード ジャパン <https://www.buzzfeed.com/jp/harunayamazaki/satotakeru-line-0214>

^{*5} 本書が出る技術書典 14 の開催期間 (2023 年 5 月 20 日から 2023 年 6 月 4 日) の途中でちょうどプランが変わるので、ここでは分かりやすく、「今日は 2023 年 6 月 1 日以降だ!」という認知で進みます。2023 年 5 月時点で本書を読んでいるみなさん、いいですか、いまは 2023 年 6 月 1 日以降です。もうプランは変わったのです。

わり、それぞれのプランで送れる無料メッセージの通数も変わりました。^{*6}

たとえば友だちが 100 人いる LINE 公式アカウントから、すべての友だちに一斉送信する形でメッセージを 2 回送ったら、100 人×2 通でそれだけで 200 通になりますね。こんなふうにかくさんの友だちにメッセージを継続して送ろうとすると、有料のプランを使う必要がありますが、逆に友だちが自分ひとりだけなら 200 回送れます。なので本書でちょっと試してみる程度なら無料のコミュニケーションプランで問題ありません。

無料メッセージの通数を使い切れれば、次の月初までメッセージが送れなくなるだけで、自分で有料のライトプランやスタンダードプランに申し込まない限りは、メッセージの送りすぎで勝手に課金が始まることはありません。クレジットカードの登録も不要ですので安心してください。

1.3 LINE 公式アカウントを作る

LINE 公式アカウントについて色々説明しましたが、実際に手を動かしてやってみるのが理解への一番の近道です。あなたも LINE 公式アカウントを作ってみましょう。

LINE 公式アカウントを作る方法には、次の 4 つがあります。どれで作っても同じ「LINE 公式アカウント」ができます。

1. LINE で作る
2. LINE 公式アカウント管理アプリ^{*7}で作る
3. LINE Official Account Manager^{*8}で作る
4. LINE Developers コンソール^{*9}で作る

方法は色々ありますが、注目してほしいのは 1 番目！ そう、なんと専用のアプリや管理画面を使わなくても、作るだけならお手持ちのスマートフォンに入っている LINE で作れるんです！ 早速、LINE で LINE 公式アカウントを作ってみましょう。

^{*6} 【重要：再掲】LINE 公式アカウントの月額プラン改定に伴うプラン料金の日割り廃止のお知らせ | LINE for Business <https://www.linebiz.com/jp/news/20230125/>

^{*7} LINE 公式アカウント管理アプリについては、「1.4.1 管理アプリを使って LINE 公式アカウントからメッセージを送ってみる」で後述します。

^{*8} LINE Official Account Manager については、「2.3.1 LINE Official Account Manager にログインする」で後述します。

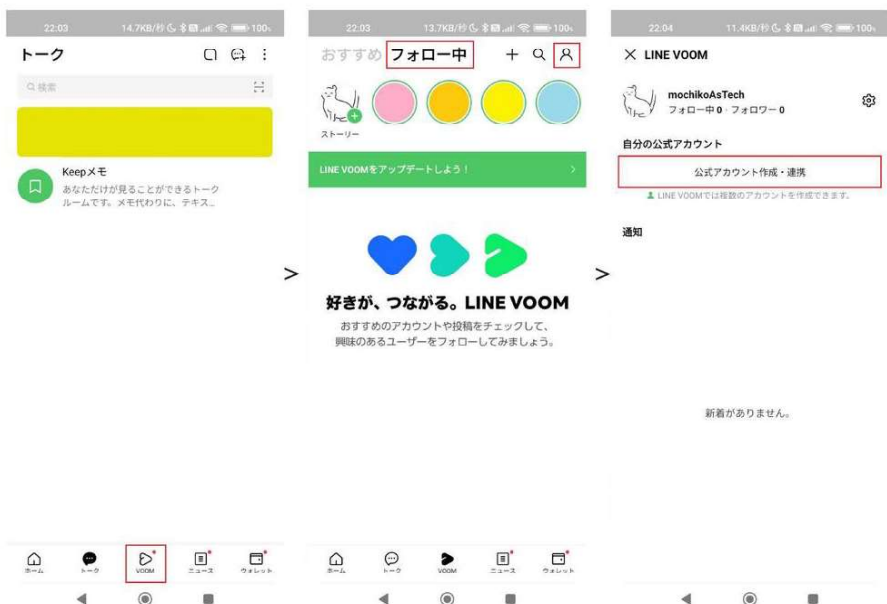
^{*9} LINE Developers コンソールについては、「2.4.1 LINE Developers コンソールでチャネルアクセストークンを発行する」で後述します。

1.3.1 LINE で LINE 公式アカウントを作ってみる

まずは LINE を開いて、下に並んでいるホーム、トーク、VOOM、ニュース、ウォレットという5つのタブのうち、中央にある「VOOM」タブをタップしてください。(図 1.3)

VOOM とやらがなんなのかということは、ひとまず考えずに進みましょう。VOOM タブを開くといきなりショート動画が再生されてびびりますが、「フォロー中」を開いた後、右上にある人型のアイコンをタップします。

そして「自分の公式アカウント」の下にある「公式アカウント作成・連携」というボタンをタップしてください。^{*10}

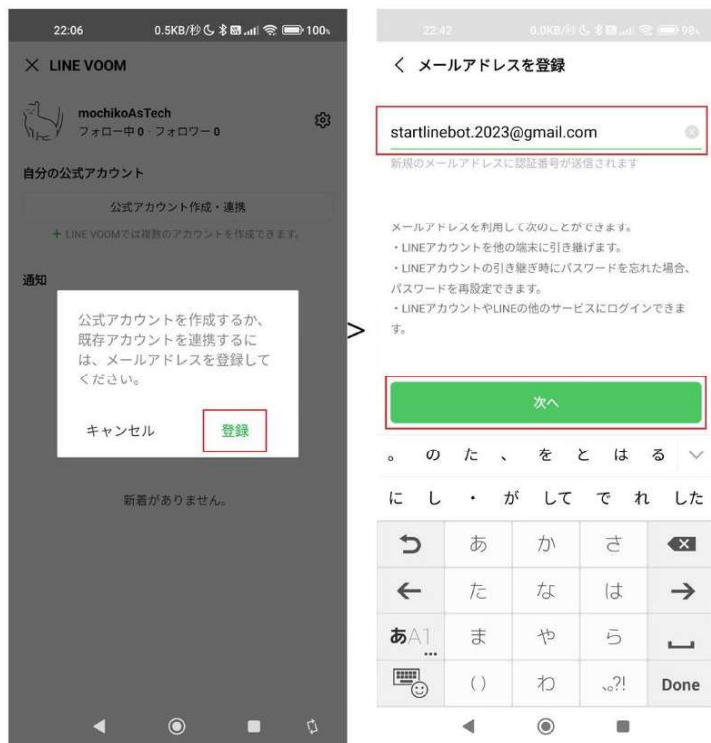


▲図 1.3 「VOOM」タブを開いて「公式アカウント作成・連携」をタップ

^{*10} 「はじめに」で「プログラム書いて動かしてみようぜ!」と元氣よく誘ってはじまった割に、ここからしばらくスマートフォンの画面でぼちぼちする作業が続きます。「俺は開発をしに來たのになんでここで焼きそばを作らされているんだ…」みたいな気持ちになるかもしれませんが、第1章でぬくもりを感じる手作業でのメッセージ配信を体験しておくことで、第2章でチャットボットに何をさせているのかが理解しやすくなるはずです。第1章の間は前向きな気持ちで焼きそば、もといスマートフォンでの画面操作をお楽しみください。

このとき、もし「公式アカウントを作成するか、既存アカウントを連携するには、メールアドレスを登録してください」と表示されたら、あなたはまだ LINE アカウントでメールアドレスを登録していないようです。その場合は、そのまま「登録」をタップします。(図 1.4)

LINE アカウントに登録するメールアドレスを入力したら、「次へ」をタップしてください。ここで登録したメールアドレスは、後でまた必要になります。どのメールアドレスを登録したのか忘れないようにしてください。



▲図 1.4 未登録だった場合はメールアドレスを入力して「次へ」をタップ

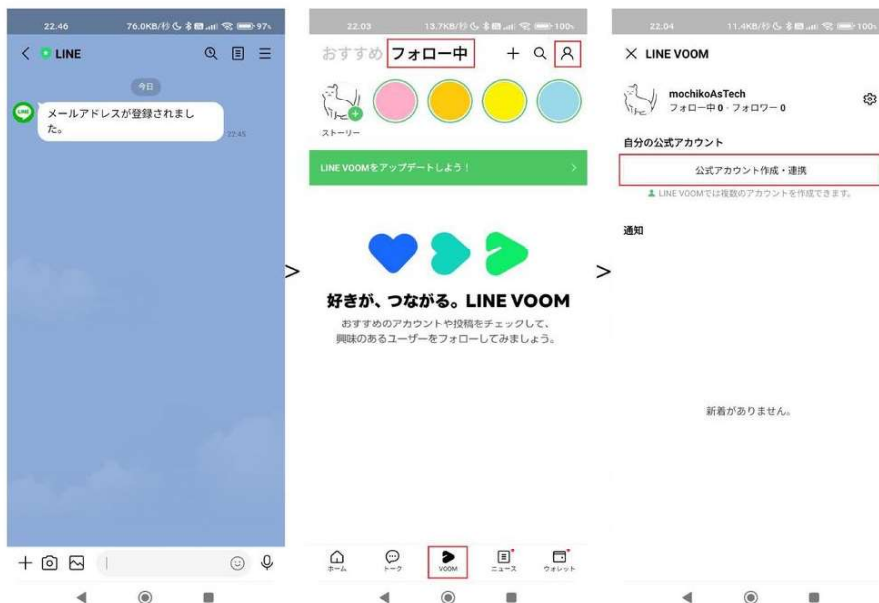
入力したメールアドレス宛てに「[LINE] メールアドレス登録確認メール」という件名のメールが届きます。(図 1.5)

メールの本文に書いてある 4 桁の認証番号を、LINE の「メール認証」の画面で入力してください。



▲ 図 1.5 メールアドレス登録確認メールが届いたら認証番号を LINE で入力

LINE に「メールアドレスが登録されました。」というメッセージが届いたら、メールアドレスの登録^{*11}は完了です。それでは改めて「VOOM」タブを開いて、右上にある人型のアイコンから、「公式アカウント作成・連携」というボタンをタップしてください。(図 1.6)



▲図 1.6 LINE にメッセージが届いたら再び「公式アカウント作成・連携」をタップ

^{*11} 登録したメールアドレスを後で変更したくなったら、下に並んでいる「ホーム」タブから右上の歯車アイコンをタップして「設定」を開き、「アカウント」の「メールアドレス」で変更できます。

LINE 公式アカウントを紹介する画面が表示されたら、「公式アカウントを作成」をタップしてください。まずは LINE 公式アカウントのプロフィール画像とアカウント名を設定します。アカウント名と言われてもピンとこないかもしれませんが、もしあなたが技術書典に出展しているなら、サークルの名前とアイコンを設定してサークルの宣伝をする LINE 公式アカウントにしてもいいでしょう。あるいは自分のハンドルネームにしてもいいと思います。このアカウント名はユニークではないため、既存の LINE 公式アカウントとかぶる「テストアカウント」のような名称であっても付けられます。^{*12} (図 1.7)

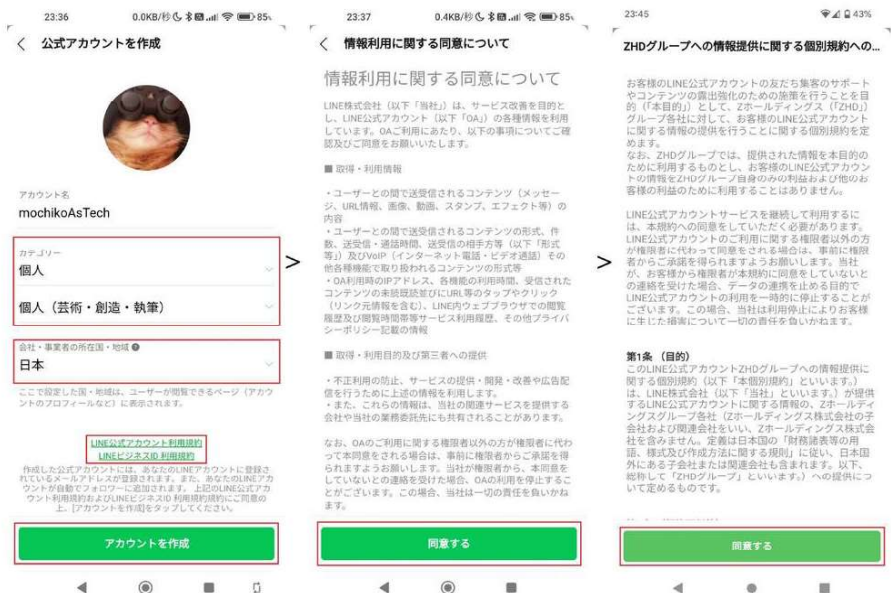


▲図 1.7 「公式アカウントを作成」からプロフィール画像とアカウント名を設定

^{*12} この後、実際にメッセージを送ってみたときにうれしくなるので、何かしら愛着の持てる名前を付けておくことをお勧めしますが、プロフィール画像もアカウント名も後から変更が可能ですので、悩みすぎずに取りあえずの画像や名前を設定しても構いません。

カテゴリーは大業種を選ぶと、その下の小業種が選べるようになります。たとえば大業種で「個人」を選ぶと、小業種に「個人（芸術・創造・執筆）」が出てきます。もし自分の本の告知用という位置づけで作るなら、この「個人（芸術・創造・執筆）」という小業種がよさそうですが、他にも「個人（IT・コンピュータ）」や「個人（学生）」、「個人（その他）」などもあるので、個人的に開発を試す用途など、LINE 公式アカウントの使い道に応じて適していると思われる業種を選択してください。[会社・事業者の所在国・地域] は、個人の場合は店舗や居住地を選択すればよいので、もしあなたが日本にお住まいなら「日本」を選んでください。（図 1.8）

下部に表示されている「LINE 公式アカウント利用規約」と「LINE ビジネス ID 利用規約」も確認したら*13、「アカウントを作成」をタップします。「情報利用に関する同意について」と「ZHD グループへの情報提供に関する個別規約への同意について」もそれぞれ確認し、同意できる内容であれば「同意する」をタップしてください。



▲図 1.8 大業種と小業種、所在国を選んだら「アカウントを作成」をタップ

*13 アカウント名などを全部入力してから利用規約を開いて読み、Android の戻るボタンで戻ったら、入力した項目が全部消えていた。つらい。

おめでとうございます！これで自分の LINE 公式アカウントができました！^{*14}（図 1.9）



▲図 1.9 LINE 公式アカウントができた！

「コンテンツを投稿して LINE VOOM をはじめよう！」とあるので、どうやらこれで、いま作った LINE 公式アカウントから LINE VOOM に動画が投稿できるようです。

でも、もともとショート動画が投稿したかった訳ではないので、それよりも友だちへのメッセージの送信を試してみたいですね。LINE 公式アカウントから友だちにトークでメッセージが送りたいときはどうすればいいのでしょうか？

^{*14} 「細かいことはいいから！早く LINE Bot を作らせて！」という方は、第 1 章の残りはスキップして第 2 章「Messaging API で LINE Bot をつくってみよう」に進んでも構いません。

1.4 LINE 公式アカウントからメッセージを送る

LINE 公式アカウントから友だちにメッセージを送る方法には、次の 3 つ^{*15}があります。

1. LINE 公式アカウント管理アプリで送る
2. LINE Official Account Manager で送る
3. Messaging API で送る

1 と 2 はどちらも LINE 公式アカウントを運用していくための管理画面で、1 がスマートフォンのアプリ、2 がウェブサイトです。3 については第 2 章「Messaging API で LINE Bot をつくってみよう」で詳しく説明しますので、ここでは「なるほど、Messaging API というものがあるのか」と軽く流してしまって構いません。

LINE 公式アカウントから友だちにメッセージを送る方法はこの 3 つですが、これらの大きな違いは自由度です。自由度は $1 < 2 < 3$ の順で高くなっていくと思ってください。1 はスマートフォンさえあれば簡単にメッセージが送れますが、細かい設定はできません。2 は、1 に比べるともう少し凝ったメッセージが作れますが、やはり細かい部分で自由度に限りがあります。そして 3 だと自分でコードを書かなければいけないですが、その分かなり自由にメッセージを作り込めます。

初めてのメッセージ送信なので、先ずはいちばんお手軽な 1 の LINE 公式アカウント管理アプリから送ってみましょう。

^{*15} LINE では、LINE 公式アカウントを作ることはできても、LINE 公式アカウントからメッセージを送るところまではできません。

1.4.1 管理アプリを使って LINE 公式アカウントからメッセージを送ってみる

いま作ったばかりの LINE 公式アカウントから友だちにメッセージを送るため、LINE を使っているスマートフォンに、LINE 公式アカウント管理アプリ^{*16}をインストールします。



▲図 1.10 LINE 公式アカウント管理アプリをインストールしよう

^{*16} Android なら <https://play.google.com/store/apps/details?id=com.linecorp.lineoa>、iPhone なら <https://apps.apple.com/jp/app/id1450599059> からインストールできます。

インストールできたら LINE 公式アカウント管理アプリを開いてください。(図 1.11)



▲図 1.11 LINE 公式アカウント管理アプリを開く

[LINE アプリで登録・ログイン] をタップします。(図 1.12)

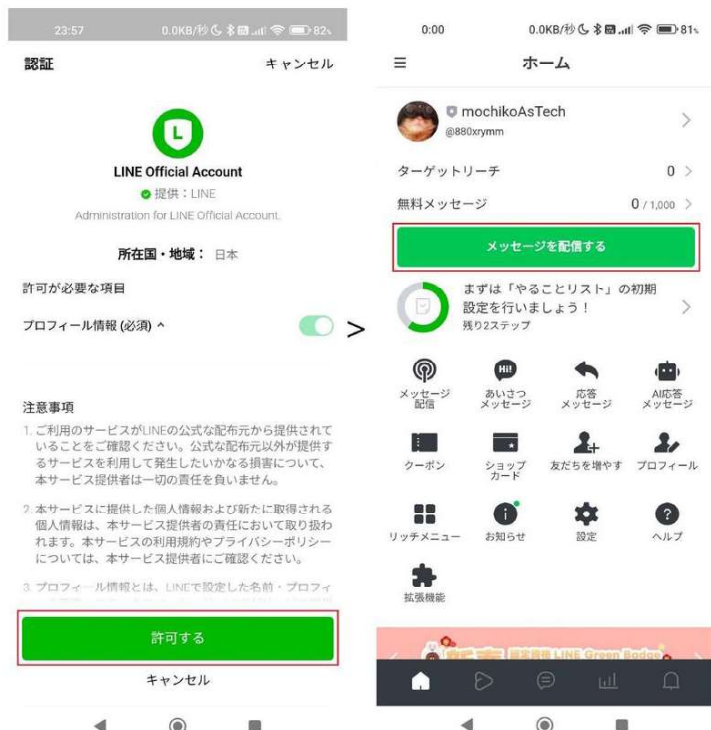
先ほど LINE 公式アカウントを作成するときに確認を求められた、[LINE ビジネス ID 利用規約] への同意が再び求められますので、[ログイン・登録] をタップしてください。



▲図 1.12 [LINE アプリで登録・ログイン] と [ログイン・登録] をタップ

すると、あなたが LINE で登録しているプロフィール情報を LINE 公式アカウントの管理アプリに提供して良いか、という認証画面が表示されます。問題なければ「許可する」をタップしてください*17。(図 1.13)

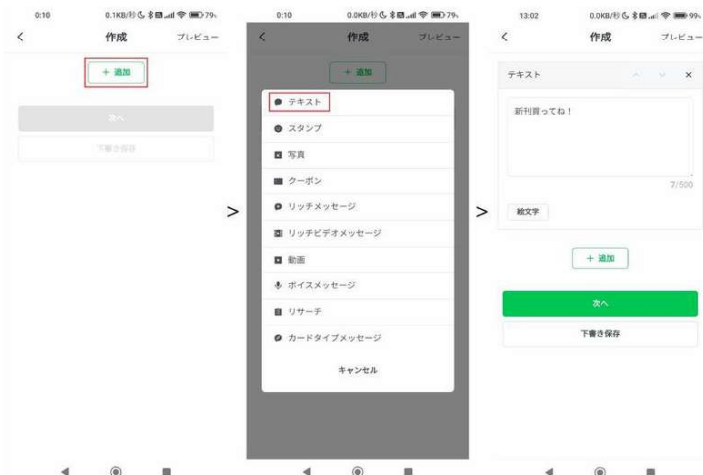
管理アプリへにログインすると、さっき作った LINE 公式アカウントが表示されます。早速、「メッセージを配信する」をタップして、メッセージを送ってみましょう。



▲図 1.13 「許可する」で LINE 公式アカウントが表示されたら「メッセージを配信する」

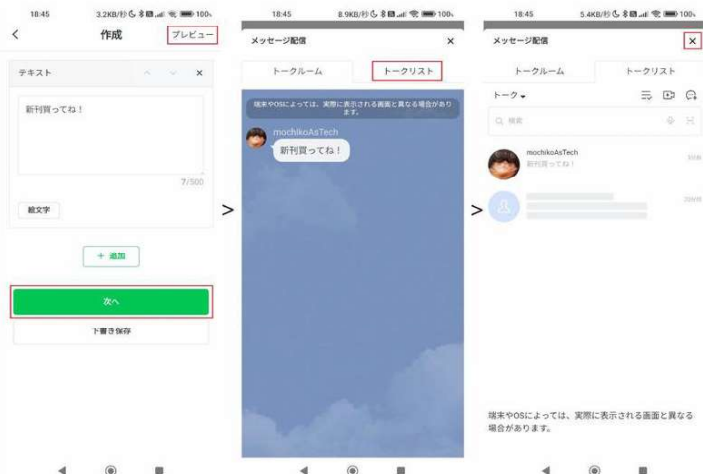
「追加」をタップして「テキスト」を選び、適当なテキストを入力します。(図 1.14)

*17 もしここで許可した情報提供を、後で取り消したくなった場合は、LINE の「設定」から「アカウント」を開いて、「連動アプリ」で確認したり、取り消したりできます。 <https://developers.line.biz/ja/docs/line-login/managing-authorized-apps/>



▲図 1.14 「追加」 から「テキスト」を選んでテキストを入力する

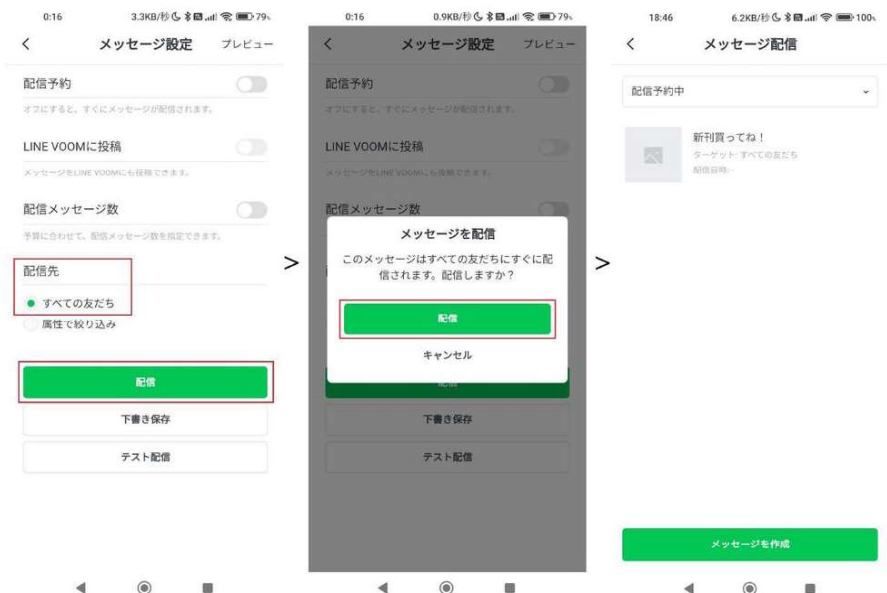
これでどんなメッセージが送られるのか、右上の「プレビュー」から確認してみましょう。(図 1.15)



▲図 1.15 右上の「プレビュー」からトークルームやトークリストでの見え方を確認しよう

「トークルーム」では、こんな感じの見た目になるようです。「トークリスト」をタップすると、トークの一覧画面での見え方も確認できます。見え方を確認したら、プレビューは右上の「×」で閉じて「次へ」をタップします。

配信先が「すべての友だち」になっていることを確認し、「配信」をタップしてメッセージを送ってみましょう。「このメッセージはすべての友だちにすぐに配信されます。配信しますか？」という確認が表示されました。さてここで本当に「すべての友だち」に送ったら、LINE の友だち全員に新刊の購入を迫る変なメッセージが届いてしまわないか、ちょっと不安になりますね。(図 1.16)



▲図 1.16 「すべての友だち」に「配信」してみよう

でも大丈夫です。あなたは LINE から、LINE 公式アカウントを作りましたが、LINE アカウントの友だちと、LINE 公式アカウントの友だちは別物なので、LINE で友だちになっている人全員にこのメッセージが届いてしまうことはありません。安心して「配信」をタップしましょう。

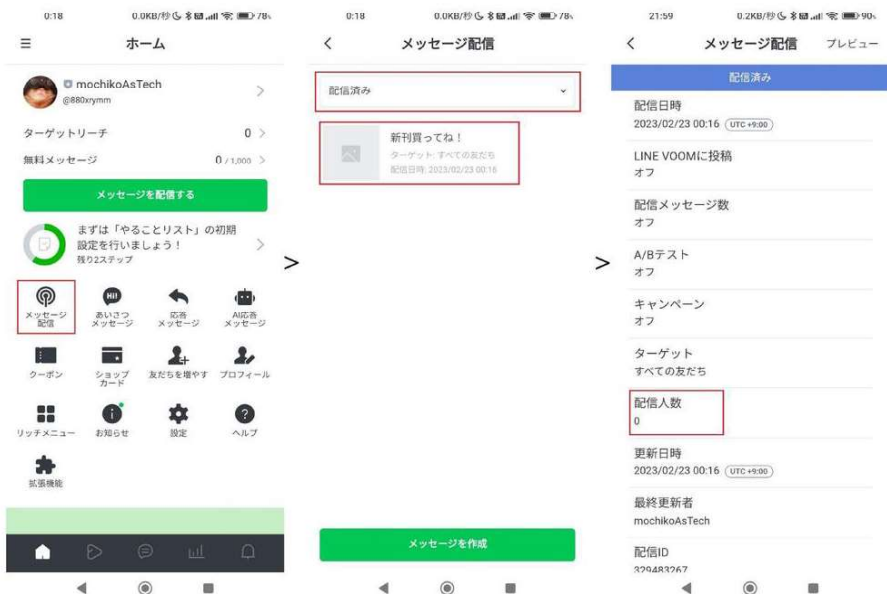
では自分の LINE に戻って、メッセージが届いたか確認してみましょう。友だちみんなには届かなくても、少なくともこの LINE 公式アカウントの創造主である自分にはメッセージが届いているかな、と思いましたが、残念ながら何も届いていません。メッセージを送信した履歴も残っていません。(図 1.17)



▲図 1.17 自分にもメッセージは届いていなかった

トークルームに送信したメッセージが残っていないのは、前述のとおり LINE アカウントと LINE 公式アカウントが別物だからです。先ほどのメッセージは、LINE 公式アカウントから送ったのであって、あなたの LINE アカウントから送った訳ではないので、LINE のトークルームに送信履歴は残っていません。そしてメッセージを受信していないのは、あなたがまだ、自分で作った LINE 公式アカウントと友だちになっていないからです。

管理アプリの「メッセージ配信」から「配信済み」を開いて、いま送ったメッセージを確認しても「配信人数」は0です。なぜならば友だちがひとりもない*18から！（図 1.18）



▲図 1.18 送ったメッセージの「配信人数」が0になっている

*18 LINE 公式アカウントに友だちがいない、という話なので、自分がダメージを受ける理由はないんだけど、どうしても「友だちがひとりもない」という響きが暴力的で、なんかこう「違うんです違うんです！」と言いつつをたくなってしまう。ちなみに図 1.17 のトークリストにも友だちはいませんが、これは本書の検証用に使っているサブのスマートフォンの LINE アカウントだからです。いるよ！！友だち！！

【コラム】個人の LINE アカウントを LINE 公式アカウントに切り替えられる？

たとえばあなたがおしゃれなタピオカミルクティー屋さんの店員で、TikTok などたくさんフォロワーを抱えるインフルエンサーでもあり、LINE の友だちも 3,000 人くらいいるとします。^{*19}

折角友だちがいっぱいいるので、TikTok で認証済みアカウントになって認証バッジをもらうような感じで、個人の LINE アカウントを運営に認証してもらって LINE 公式アカウントに切り替えられないのかな、と思ったのですが、そんなことは可能なのでしょうか？

結論から言うと、個人の LINE アカウントを LINE 公式アカウントに昇格させることはできません。折角個人の LINE アカウントでたくさんの友だちを集めても、それをそのまま LINE 公式アカウントに引き継ぐことはできないので、仕事や宣伝を目的として LINE で友だちを増やすなら、最初から LINE 公式アカウントではじめるのがお勧めです。

^{*19} 普段、Twitter で暮らしているので、おしゃれなインフルエンサーの解像度が低い。

1.4.2 LINE 公式アカウントと友だちになる

残念ながら友だちがひとりもない状態でメッセージを配信してしまったため、折角送ったメッセージですが誰にも届かなかったようです。LINE 公式アカウントからメッセージを送って、ちゃんと「友だち」に届くことを確認するため、まずは自分が LINE 公式アカウントと友だちになってみましょう。

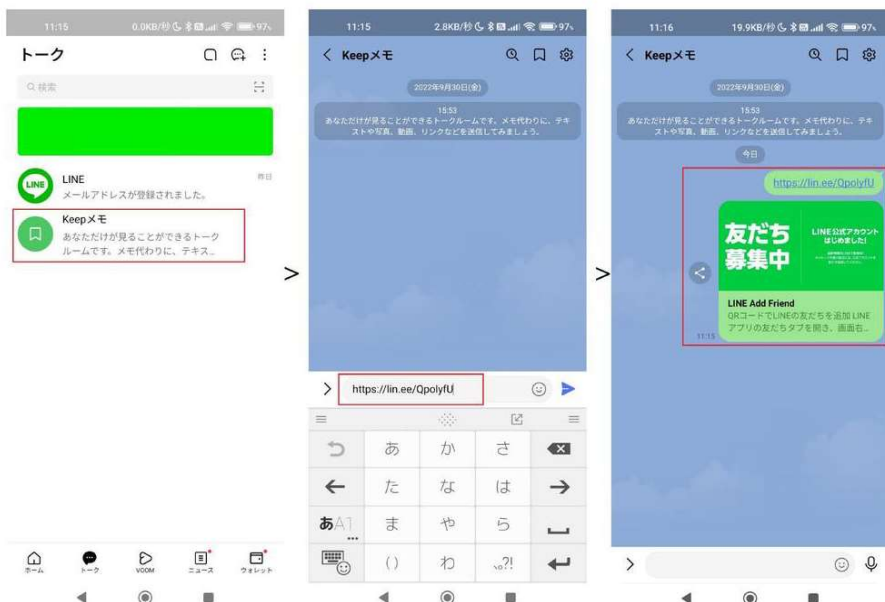
LINE 公式アカウントの管理アプリで「友だちを増やす」をタップして、「URL を作成」を開きます。「URL をコピー」をタップして、友だち追加用の URL^{*20}をコピーしましょう。（図 1.19）



▲図 1.19 「友だちを増やす」から「URL を作成」を開いて「URL をコピー」する

^{*20} ちなみに本書で筆者が作ったこの LINE 公式アカウントとは実際に友だちになれます。LINE で筆者と友だちになってみよう。 <https://lin.ee/Qp0lyfU>

LINE に戻って、Keep メモという自分専用トークルームみたいなところに友だち追加の URL をペーストします。元気よく「友だち募集中」と表示されました。タップして友だちになってみましょう。(図 1.20)

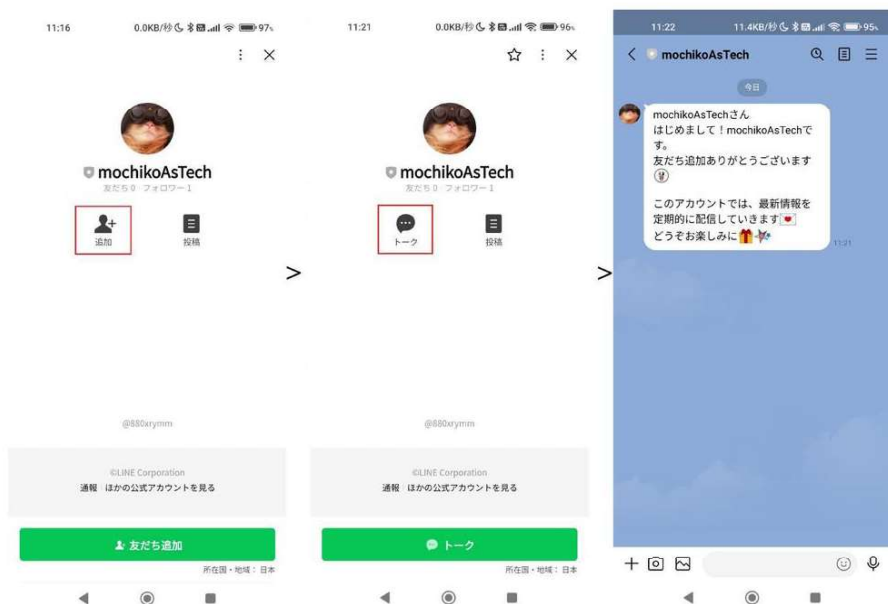


▲図 1.20 LINE の Keep メモに URL をペーストする

この友だち追加用の URL をうっかり Twitter などを書いてしまうと、誰でもあなたの LINE 公式アカウントと友だちになってしまいます。自分しかメッセージを受け取っていないつもりで色々テストしていたら、実は自分以外の人もひっそり見ていた、ということにならないよう、友だち追加の URL の取り扱いには注意しましょう。

URLを開くと、先ほど作った LINE 公式アカウントの友だち追加画面^{*21}が表示されました。さっそく「追加」のアイコンをタップして、LINE 公式アカウントと友だちになってみます。友だちになると、アイコンが「追加」から「トーク」に変わるので、「トーク」をタップしてトークルームを開いてみましょう。(図 1.21)

設定した覚えもないのに「はじめまして！ mochikoAsTech です。友だち追加ありがとうございます」という元気なメッセージが届いています。誰だお前、私が作った LINE 公式アカウントだけど誰だ。これは LINE 公式アカウントの「あいさつメッセージ^{*22}」という機能で、友だち追加されたときに、自動で任意のメッセージを送ることができます。LINE 公式アカウントを作ると、デフォルトで「あいさつメッセージ」が設定されているので、この元気なメッセージが届いたという訳です。



▲図 1.21 「追加」をタップすると友だち追加されて「トーク」に変わる

^{*21} 「友だち 0」[フォロワー 1]とありますが、このフォロワーというのは LINE VOOM で LINE 公式アカウントをフォローしている人のことです。さっき LINE 公式アカウントを作ったときに、自動的に自分で自分をフォローしていたので、この「フォロワー 1」は自分自身です。知らない誰かではありません。

^{*22} あいさつメッセージについては、「2.6.4 友だち追加されたときのあいさつメッセージを併用する」でも後述します。

【コラム】 開発用途の LINE 公式アカウントが知らない人に友だち追加されてしまった。ブロックできる？

個人的な開発用途として LINE 公式アカウントを作っていたのに、うっかり友だち追加の URL を Twitter に書いてしまい、知らない人に友だち追加されてしまいました。あくまで開発用途の LINE 公式アカウントなので、他人にあれこれ見られたくない…そんなとき、LINE 公式アカウント側から友だちをブロックすることはできるのでしょうか？

結論から言うと、LINE 公式アカウント側から友だちをブロックしたり、削除したりすることはできません。仮に友だち追加した人を特定して、お願いして目の前でブロックしてもらったとしても、後でこっそりブロック解除されることは止められません。一度友だち追加されてしまったら、もう友だちが自分だけだった元の状態には戻せないと思っておいたほうがいいでしょう。

自分以外とは誰とも友だちになっていない状態に戻したいのであれば、友だち追加されてしまった LINE 公式アカウントは削除して、新しい LINE 公式アカウントを作り直しましょう。LINE 公式アカウントは、LINE Official Account Manager、または LINE 公式アカウント管理アプリの「アカウント設定」から、「アカウントを削除」で削除できます^{*23}。LINE 公式アカウントは、一度削除してしまうと復活させることはできないので注意してください。

^{*23} アカウントを削除するには？ | LINE for Business <https://linebiz.force.com/help/s/article/000001102?language=ja>

1.4.3 LINE 公式アカウントから友だちにメッセージを送ってみる

再び LINE 公式アカウント管理アプリに戻ると、「ターゲットリーチ」が先ほどの 0 から 1 に変わっています！（図 1.22）

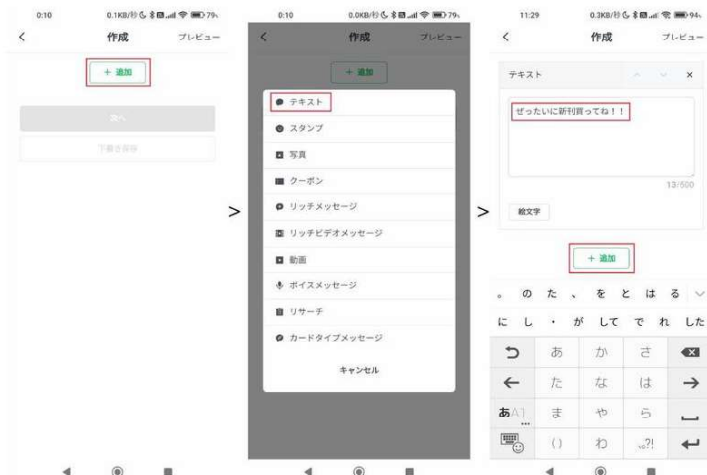


▲図 1.22 ターゲットリーチが 0 から 1 になった

ターゲットリーチとは、要は「送信対象となる友だち」の総数です。友だちできた…ひとりできた…自分だけど…。鏡に向かって「私たち今日から友だちだね…！」*²⁴と言っている気分になってきましたが、例え相手が自分自身だろうが友だちは友だちです！メッセージを送ってみましょう。先ほどと同じように「メッセージを配信する」をタップします。

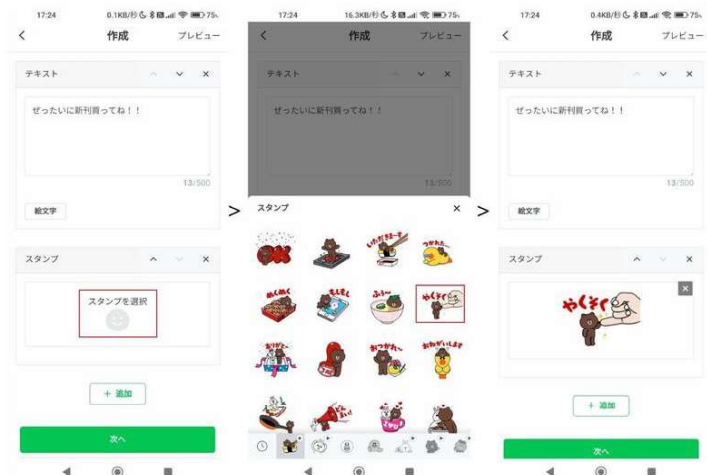
「追加」をタップして、まずは「テキスト」を選び、適当なテキストを入力します。（図 1.23）

*²⁴ 食器を仕舞った本棚のガラス戸に映った自分をケティ・モリスと呼んで仲良くしています。早くダイアナに出会いたいですね。



▲図 1.23 「追加」から「テキスト」を選んでテキストを入力する

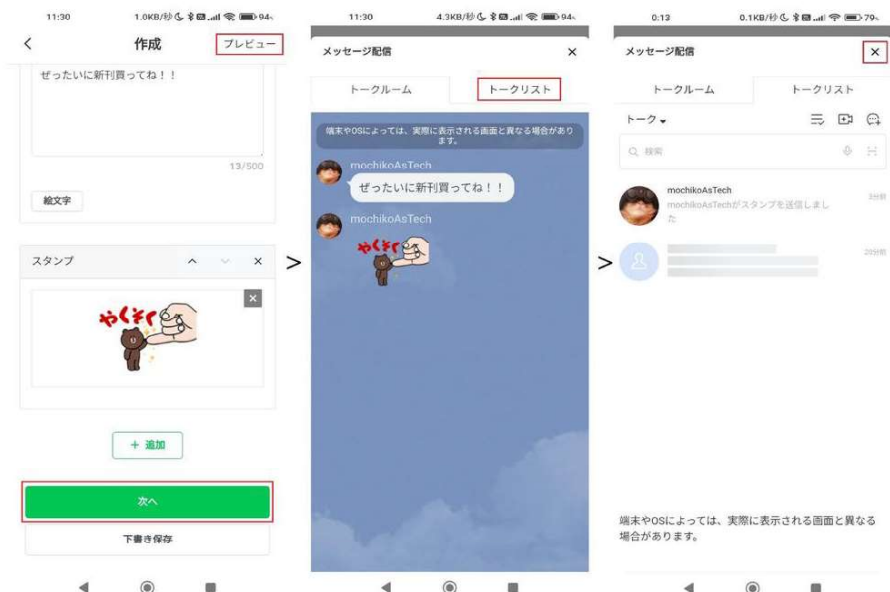
さらに「追加」をタップして、今度は「スタンプ」を選び、「スタンプを選択」から適当なスタンプを選びます。(図 1.24)



▲図 1.24 「追加」して適当なスタンプを選択する

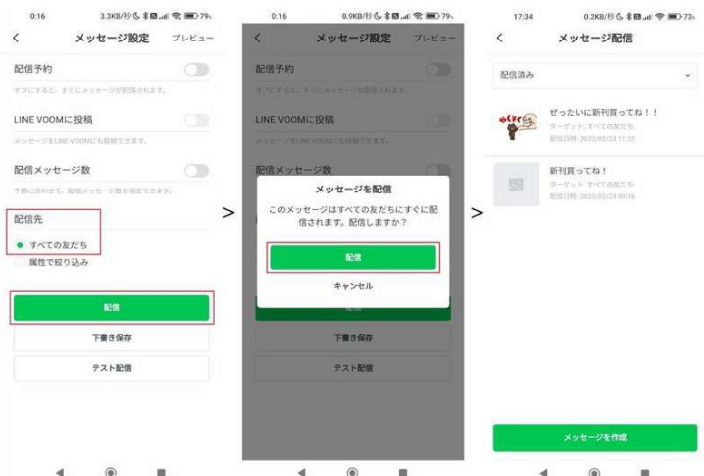
これでどんなメッセージが送られるのか、右上の「プレビュー」から確認してみましょう。(図 1.25)

「トークルーム」では、こんな感じの見え目になるようです。「トークリスト」をタップすると、トークの一覧画面での見え方も確認できます。さっきより念押しが強めのメッセージが準備できました。見え方を確認したら、いったんプレビューは右上の「×」で閉じましょう。



▲図 1.25 右上の「プレビュー」からトークルームやトークリストでの見え方を確認しよう

「次へ」をタップして、そのまま「すべての友だち」に送ってみましょう。さっきは誰にも届きませんでした、いまはたったひとりの友だちに届くはず。私に届け！ 私のメッセージ…！ 祈りを込めて「配信」をタップします。(図 1.26)



▲図 1.26 「すべての友だち」にメッセージを「配信」する

ピロン！ LINE の通知が来ました！ 新刊の購入をお願いするメッセージが届いたようです。(図 1.27)



▲図 1.27 LINE 公式アカウントからメッセージが届いた！

やったね！これで「LINE 公式アカウントから友だちにメッセージを送る」というアチーブメントを達成しました。

【コラム】LINE 公式アカウントのメッセージはさかのぼって見られる？

LINE 公式アカウントと友だちになると、メッセージが届くようになりますが、友だちになる以前のメッセージはさかのぼって見られるのでしょうか？

残念ながらメッセージが届くのは友だちになってからなので、友だちになる以前に送信されていた LINE 公式アカウントのメッセージを、Twitter や Instagram のようにさかのぼって見ることはできません。

LINE 公式アカウントから友だちへメッセージを送るときに、同内容を LINE VOOM にも投稿していれば、そちらはさかのぼって見られます。

【コラム】メッセージの通数はどうカウントされるのか

たとえば1人の友だちに対してLINE 公式アカウントから、こんなふうテキストと画像とスタンプという3つのメッセージオブジェクト*25が含まれるメッセージを送った場合、メッセージの通数は1通でしょうか？ それとも3通でしょうか？（図 1.28）



▲図 1.28 テキストと画像とスタンプのメッセージ

メッセージの通数は送信対象となった友だちの人数でカウントされるので、このように3つのメッセージオブジェクトをまとめて送った場合でも、カウントは1通となります。もしテキストを送る、画像を送る、スタンプを送る、というようにメッセージを3回に分けて送ったらカウントは3通ですね。ちなみに1つの

メッセージにつきメッセージオブジェクトは LINE Official Account Manager や管理アプリでは最大 3 つ、Messaging API では最大 5 つまで指定できます。

メッセージは友だちと LINE 公式アカウントの 1 対 1 のトークだけでなく、何人もの友だちが参加しているグループトークに対しても送信可能です。前述のとおり、メッセージの通数は送信対象となった友だちの人数でカウントされるので、友だち 3 人と LINE 公式アカウントが参加しているグループトークに、LINE 公式アカウントからメッセージを送った場合、メッセージの通数は 3 通としてカウントされます。

なおどんなメッセージでも無料メッセージの通数を消費する訳ではありません。友だち追加されたときのあいさつメッセージや、Webhook^{*26}に含まれる応答トークンを使って送る応答メッセージなどは、何通送っても通数としてカウントされません。^{*27}

1.4.4 LINE 公式アカウントにメッセージを送ってみる

LINE 公式アカウントからメッセージで「絶対に新刊買ってね!」と頼まれたので、ここで「新刊買います!」と返事をしてあげたいのですが、LINE 公式アカウントに返事を送ったらいったいどうなるのでしょうか? 物は試しです。LINE で、LINE 公式アカウントに対してメッセージを送ってみましょう。

LINE 公式アカウントに「新刊買います!」とメッセージを送ってみたところ、なんとまたしても設定した覚えのない「申し訳ありませんが、このアカウントでは個別のお問い合わせを受け付けておりません」というメッセージが返ってきました。誰だお前! 私が作った LINE 公式アカウントで勝手に返事してきて誰なんだ。(図 1.29)

^{*25} 「なんかさざっと出てきたけどメッセージオブジェクトってなんなんですか? わたくし分かりませんわ!」「遠方のご学友にお手紙を出すときのことを想像なさって。封筒にお気持ちをしたためた便箋を入れて、お茶会でお撮りあそばした思い出の写真も入れて、贈り物のぶくぶくファンシーシールも入れますでしょ? その便箋や写真やシール、ひとつひとつがメッセージオブジェクトなんですわ!」「なるほどですわ! 便箋と写真とシールは、テキストと画像とスタンプの比喻ですわね! ところでどうしてわたくしたち、突然お嬢様なんですか?」「説明しにくいことはお嬢様になるとなぜかうまいこと説明できるんですわ!」

^{*26} Webhook や応答トークンについては、「2.5.5 Webhook とは」で後述します。

^{*27} 課金対象となるメッセージについて | LINE for Business <https://www.linebiz.com/jp/service/line-official-account/plan/>



▲図 1.29 メッセージを送ったら「応答メッセージ」が届いた

これは LINE 公式アカウントの「応答メッセージ*²⁸」という機能で、ユーザーからメッセージが届いたときに、自動で任意の返答を送ることができます。LINE 公式アカウントを作ると、デフォルトで「応答メッセージ」が設定されているので、このメッセージが届いたという訳です。

*²⁸ 応答メッセージについては、「2.5.1 方法 1. 固定の自動応答を設定しておいて個別対応は一切しない」で後述します。

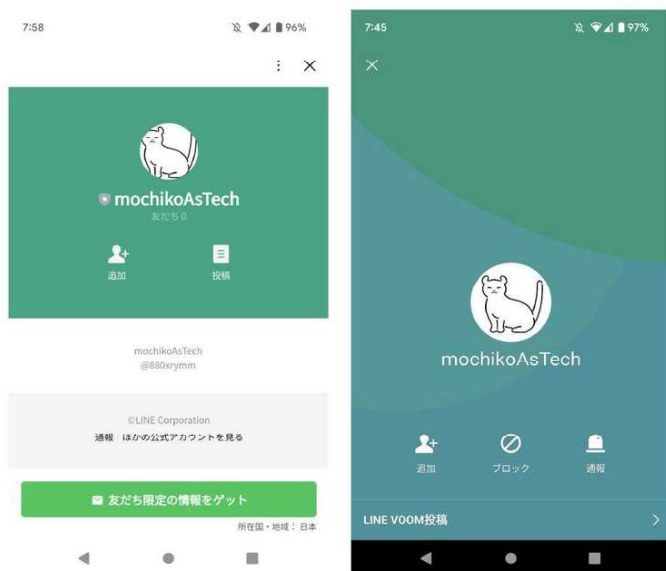
1.5 LINE 公式アカウントの基礎知識

無事に LINE 公式アカウントから友だちへのメッセージ送信もできたので、ここでちょっと LINE 公式アカウントについていくつか説明をしておきましょう。実践の後の座学タイムです。

1.5.1 LINE 公式アカウントと普通の LINE アカウントとの違い

先ほど LINE アカウントと LINE 公式アカウントは別物という話をしましたが、普通の LINE アカウントと LINE 公式アカウントはどうやったら見分けをつけられるのでしょうか？ LINE 公式アカウントを運用している本人にはもちろん分かりますが、どちらも QR コードを読み込んで友だち追加できるので、友だちになる側はどこで「あ、これ LINE 公式アカウントだ」と気づけるのでしょうか？

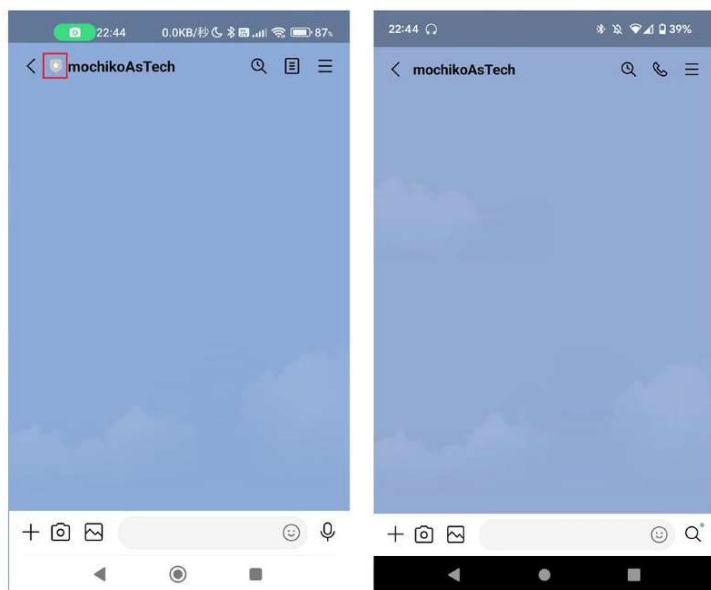
友だち追加の画面を比較してみましょう。左が LINE 公式アカウントで、右が普通の LINE アカウントです。（図 1.30）



▲図 1.30 左が LINE 公式アカウントで、右が普通の LINE アカウント

LINE 公式アカウントの友だち追加画面には追加、投稿のボタンが並んでいますが、普通の LINE アカウントの友だち追加画面だと追加、ブロック、通報のボタンが並んでいます。また LINE 公式アカウントの方は、ベーシック ID^{*29}が表示されていたり、[ほかの公式アカウントを見る]や[友だち限定の情報をゲット]とあるので、よく見れば「これ、LINE 公式アカウントだな?」と気づけそうです。

トークルームの画面も比較してみましょう。LINE 公式アカウントは名前の左に「未認証^{*30}の LINE 公式アカウント」であることを示す灰色のバッジが付いています。(図 1.31)



▲図 1.31 LINE 公式アカウントは名前の左に灰色のバッジが付いている

普通の LINE アカウントと LINE 公式アカウントには、このような見た目の違いがあります。

^{*29} ベーシック ID については、「1.5.2 プレミアム ID とベーシック ID」で後述します。

^{*30} 未認証アカウントについては、「1.5.3 未認証アカウントと認証済アカウント」で後述します。

1.5.2 プレミアム ID とベーシック ID

LINE 公式アカウントを作ると、最初に「ベーシック ID」というものが自動で割り振られます。@マークと、ランダムな 3 桁の数字と 5 桁の英字で構成された「@880xrymm」みたいなものがベーシック ID です。

LINE の検索窓でこのベーシック ID を検索 (図 1.32) すると、ちゃんと LINE 公式アカウントにたどり着けるのですが、友だち登録してもらうためにランダムな文字列をユーザーに「最初が@で…880 の…えっくすあーる…わい…えむえむ」のように伝えて入力させるのはなかなか難儀です。もうちょっと名前っぽい、覚えやすい ID がほしくなりますよね。



▲図 1.32 友だち検索でベーシック ID を検索すると LINE 公式アカウントが表示される

そこでなんと、月額 100 円 (税別) を出せば、4 文字から 18 文字で希望の「プレミアム

ID^{*31}」というものが取得できます！ただし、他の人がすでに使っているプレミアム ID は取得できません。

プレミアム ID と有料プランは別枠なので、無料のコミュニケーションプランのまま、プレミアム ID だけ購入することも可能です。またプレミアム ID を取得しても、ベーシック ID や、プレミアム ID 取得前に作った友だち追加の URL、QR コードなどは引き続き使用できます。

1.5.3 未認証アカウントと認証済アカウント

無料プランと有料プラン、ベーシック ID とプレミアム ID についてはすでに説明しましたが、実はさらに認証ステータスというものがおり、LINE 公式アカウントは未認証アカウントと認証済アカウント^{*32}の2種類に分けられます。

認証済アカウントになるためには申請をして、審査に通過する必要がありますが、この申請と審査は無料です。有料プランやプレミアム ID と違って、認証済アカウントになるために費用はかかりません。

では認証済アカウントになると何がうれしいのでしょうか？

認証済アカウントになると、バッジの色が灰色から青色になって「公式っぽさ」が増すと共に、LINE 内でアカウント名や概要が検索対象^{*33}になり、検索結果に表示されるようになります。^{*34}逆に言えば、デフォルトの未認証アカウントであれば、友だち追加の QR コードや URL、ベーシック ID、あるいはプレミアム ID を自ら露出しない限り、知らない人から勝手に友だち追加される可能性は低いということです。

また認証済アカウントになると、「友だち募集中」と書かれたキャラクターと QR コード付きのステッカー、三角 POP などが購入できます。お店の入り口に貼ったり、テーブルに置いたりして、友だち追加を促すためのおしゃれな販促物を、わざわざ自分で作らなくても簡単に購入できるのがいいところですね。

^{*31} プレミアム ID とは | LINE for Business

<https://www.linebiz.com/jp/service/line-official-account/plan/>

^{*32} 「認証済アカウント」とは？ | メリット・申請方法 | LINE for Business <https://www.linebiz.com/jp/service/line-official-account/verified-account/>

^{*33} 未認証アカウントの場合、アカウント名や概要に含まれる単語で検索しても検索結果には出てきません。ですが、ベーシック ID やプレミアム ID で検索した場合は、「1.5.2 プレミアム ID とベーシック ID」のとおり、未認証アカウントであってもしっかり検索結果に表示されます。

^{*34} 厳密に言えば、認証済アカウントになると検索結果に表示させるのか、非表示にしておくのかを選べるようになります。たとえば「お金を払って謎解きゲームのチケットを買った人だけ、LINE 公式アカウントと友だちになって、LINE のトークからゲームに参加できる」というようなことを実現したい場合は、認証済アカウントであっても検索結果には表示されないよう「非表示」を選んでおきましょう。
<https://www.linebiz.com/jp/manual/OfficialAccountManager/tutorial-step5/>

ただし認証済アカウントの審査は、個人名では通らない^{*35}ようです。販促物が購入できる、という特典を見ても、認証済アカウントはインフルエンサーなどの個人^{*36}ではなく、店舗やサービスなどをターゲットとしているようです。

【コラム】準備中の LINE 公式アカウントをリリース日まで非公開にしておける？

準備中の LINE 公式アカウントが人目に触れてしまうことのないよう、リリース日まではアカウントそのものを非公開にしておきたい！と思ったとします。そんなことは可能なのでしょうか？

残念ながら LINE 公式アカウントには「非公開」と「公開」、あるいは「開発中」と「リリース済み」というような状態管理がありません。作られた瞬間から、ベーシック ID やプレミアム ID さえ分かれば、実態がリリース前だろうがなんだろうが誰でも友だち追加できてしまいます。

リリース前や準備用の LINE 公式アカウントは、うっかりベーシック ID やプレミアム ID、そして友だち追加の URL などを外部に露出させないように注意しましょう。また認証済アカウントにした場合は、「検索結果での表示」をデフォルトの「非表示」のままにしておきましょう。

^{*35} LINE 公式アカウントの審査とは | 認証済アカウント申請時の注意点 <https://www.linebiz.com/jp/column/technique/20190829/>

^{*36} ちなみに 2022 年 6 月には「LINE クリエイターアカウント」という、インフルエンサーやクリエイター向けの LINE 公式アカウントの新カテゴリーも発表されていました。 <https://linecorp.com/ja/pr/news/ja/2022/4265>

CHAPTER

2

Messaging APIで LINE Botをつくってみよう

LINE公式アカウントの「中の人」を人間の代わりにボットにしよう。

2.1 LINE 公式アカウントをチャットボットにしたい

第1章「LINE 公式アカウントをつくってみよう」では LINE 公式アカウントを作り、管理アプリを使って友だちにメッセージを送信しました。LINE 公式アカウント管理アプリや LINE Official Account Manager には色んな機能があるので、それらを使ってお店やサービスの LINE 公式アカウントを運用していくことも可能です。

ですが、本書では LINE 公式アカウントを手作業で運用していきたい訳ではなく、「中の人」を人間からボットに変えて、LINE 公式アカウントを自動で応答するチャットボットにしたいのです。

2.1.1 チャットボットとは

急にチャットボットという言葉が出てきましたが、チャットボットとはいったいなんでしょう？

チャットボットとは、リアルタイムにメッセージをやりとりする「チャット」と、人間のように動いたり働いたりする機械の「ロボット」を組み合わせた言葉です。チャットボットの裏側には人間がいるわけではなく、プログラムがメッセージの内容に応じた返信をしてくれたり、自動でメッセージを送ってきたりしています。

このチャットボットは、近年では身近な存在になったことで、単に「ボット」と呼ばれることの方が多いかもしれません。Twitter で地震が起きるとすぐに震度を知らせてくれる地震速報ボットや、著名人の名言を定期的につぶやくボット、特定の用語に反応してリプライしてくるボットなどを、あなたも一度は見たことがあるのではないのでしょうか。会社の Slack に、GitHub の通知をしてくれるボットや、メッセージを自動翻訳してくれるボットがいる、というケースもあるかも知れません。

本書では、この自動応答するチャットボットのプログラムのことを「ボット」と呼びます。

2.2 Messaging API でチャットボットを作るには

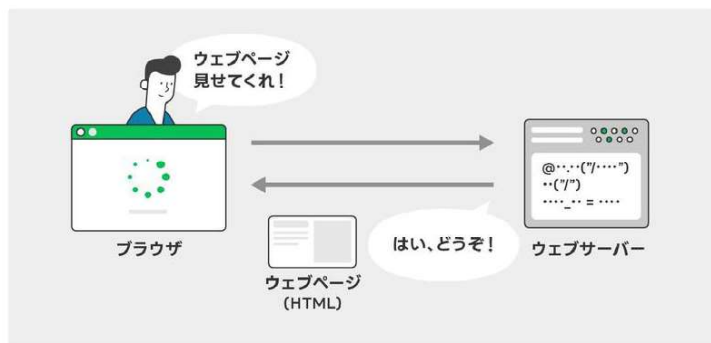
ボットを作って、LINE 公式アカウントをチャットボットにするには、LINE の Messaging API というものを使います。

2.2.1 API とは

さて、Messaging API とは何か、という話をする前に、そもそも「API」とは何か、という説明をさせてください。

Messaging API の「API」は、Application Programming Interface の略です。名前のとおり、別々のアプリケーションがお互いに情報をやりとりするときの接点となる、窓口のようなものだと思ってください。そして本来、API はとても広い意味をもつ言葉ですが、この Messaging API における API とは「REST API」*1のことだと思われます。

通常、私たちはブラウザで URL を入力したり、リンクをクリックしたりすることで「このウェブページを見せてくれ!」とウェブサーバーにリクエストを投げ、リクエストを受けたウェブサーバーが「はい、どうぞ」とウェブページをレスポンスで返してくれます。(図 2.1)



▲図 2.1 ブラウザでウェブページを見るときのリクエストとレスポンス

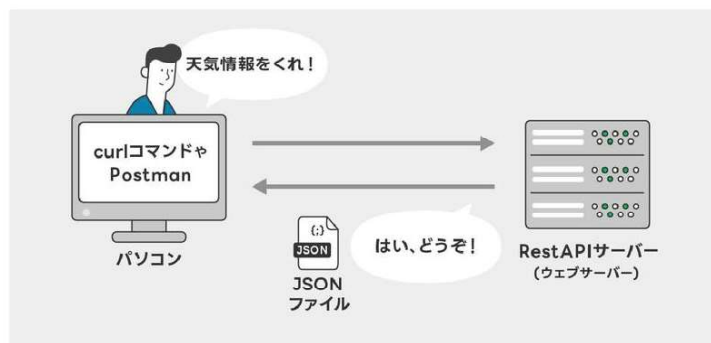
バーで「ビールをくれ!」というリクエストを投げると、ビアサーバーからのレスポンスとしてよく冷えたビールが返ってくる、みたいなものですね。*2

REST API は、このウェブページと同じように、ウェブサーバー上で提供されます。

*1 REST は REpresentational State Transfer の略。REST アーキテクチャスタイルという、ルールのようなものに従って作られた API が REST API や RESTful API と呼ばれます。

*2 シンガポールのブルバーで飲んだタイガービールがおいしかった。また行きたい。

私たちが curl コマンド^{*3}や、Postman^{*4}や、プログラムを通じて REST API に対して「天気情報をくれ!」とか「メッセージを送信してくれ!」というリクエストを投げると、ウェブサーバー上でうごく REST API が「はい、どうぞ! 君が求めていた天気情報はこれだよ!」とか「メッセージの送信に成功したよ!」というようにレスポンス^{*5}を返してくれるのです。(図 2.2)



▲図 2.2 REST API をたたくときのリクエストとレスポンス

前述のとおり、API は広義には「情報をやりとりする窓口」であり、さまざまな意味を内包していますが、本書においてはただ「API」と呼んだら、それは REST API のことを指していると思ってください。

^{*3} curl (カール) は HTTP や HTTPS、SCP、LDAP など、さまざまなプロトコルでデータ転送ができるコマンドです。今までに curl を使ったことのない人にとっては、この説明を読んでもいまいちピンとこないと思うので、「ターミナル」という種類のソフトで、ブラウザのようなことができる命令文だと思っておいってください。ちなみにターミナルは、エンジニア以外の方には、いわゆる「黒い画面」と言った方がお馴染みかもしれません。

^{*4} Postman は、GUI の画面で REST API をたたける便利なツールです。 <https://www.postman.com/>

^{*5} 最近の REST API では、レスポンスはデータが扱いやすい JSON という形式のファイルで返ってくることが多いです。実際の JSON がどんなものなのかは、「2.4.2 Messaging API でブロードキャストメッセージを送信する」で確認できます。

【コラム】API は「たたく」もの？「呼び出す」もの？

API にリクエストを投げることを「API をたたく」と表現することがあります。キーボードの Enter キーを「ツツターン！」とたたいて API を実行するイメージから来ているのか、あるいは API のドアをコンコンとたたいて「すみません、お天気情報が欲しいんですが…」と訪問するイメージなのか、由来は筆者も分かりませんが、「API をたたく」という言葉はエンジニア界隈では割と広く使われている印象です。

日本語と同じように英語でも "Hit an API" と言うのかなと思いきや、そちらは "Call an API" という表現の方が一般的なようです。"Call an API" を日本語にすると「API を呼び出す」ですが、たしかに日本語でも「API をたたく」だけでなく、この「API を呼び出す」という表現もよく見ます。こちらは「ちょっとお天気情報を持ってこっち来てもらえますか？」と電話して呼び出しているイメージでしょうか。

API を「たたく」や「呼び出す」だけでなく、パソコンが「立ち上がる」、CPU の使用率が 100% に「張り付く」、ここの処理で「引っかかって」エラーを「吐いて」「落ちてる」、プロセスを「殺す」、といった独特な表現は、知らない人からすると面白いですね。

情報の訂正時に「あー…今言ったのは嘘です。忘れてください。正しくは○○です」のように言うてしまうことはありませんか？ この言い方、筆者も普通に使っていましたが、聞き手によっては「嘘をついた」と言われると「この人、悪意を持って誤情報を伝えたって自己申告してる…？」と不思議に思われるようです。

2.2.2 Messaging API とは

ではあらためて、Messaging API について説明していきましょう。Messaging API は、LINE 公式アカウントからのメッセージ送信や、受け取ったメッセージへの返信といった操作ができる API^{*6}です。LINE 株式会社が無料^{*7}で提供しており、LINE Developers

^{*6} Messaging API は、あくまで LINE 公式アカウントの操作をするための API なので、個人の LINE アカウントで受信したメッセージを API を使って Slack に転送する、というようなことはできません。

^{*7} Messaging API の各 API をたたくこと自体に費用はかかりません。第 1 章「LINE 公式アカウントをつくってみよう」の「1.2.1 LINE 公式アカウントにかかる費用」で紹介したとおり、送れるメッセージの通数を増やすために有料のプランを契約したときに初めてお金がかかります。

コンソールと呼ばれる開発者向けの管理画面でアカウント登録をすれば誰でも利用できます。

Messaging API を使用することで、開発者はユーザーが LINE 公式アカウントに送ったメッセージを受信したり、LINE 公式アカウントから友だちに対して返信を送ったり、友だちとなったユーザーのプロフィール情報を取得したりすることができます。

実際に Messaging API を使用するためには、LINE 公式アカウントと紐づく形で Messaging API チャンネルというものを作って、API の利用に必要な「チャンネルアクセストークン」を取得する必要があります。

LINE 公式アカウントと Messaging API チャンネルの関係は、初見だとちょっと分かりにくいので、実際に Messaging API チャンネルを作る前にそこを少し説明していきます。

2.3 Messaging API チャンネルを作ろう

LINE 公式アカウントと Messaging API チャンネルは、表と裏のような存在です。LINE 公式アカウントを単体で作って、LINE Official Account Manager や管理アプリを使っている人が頑張ることもできますが、裏側に Messaging API チャンネルを紐づけて、ボットが自動で応答するように設定することも可能です。(図 2.3)



▲図 2.3 LINE 公式アカウントと Messaging API チャンネルは表と裏

あなたはさっき、LINE で LINE 公式アカウントを作ったので、いまは表の LINE 公式アカウントだけがあって、裏に控える Messaging API チャンネルはまだ存在していない状態です。

前述の LINE Developers コンソールで Messaging API チャンネルを作ると、最初から表（LINE 公式アカウント）と裏（Messaging API チャンネル）が揃った状態で作成されるのですが、先に表だけを作った場合は、LINE Official Account Manager で裏を作って紐づけてやる必要があります。

それでは LINE Official Account Manager で、あなたの LINE 公式アカウントに紐づく Messaging API チャンネルを作りましょう。

2.3.1 LINE Official Account Manager にログインする

第1章「LINE 公式アカウントをつくってみよう」では、スマートフォンで色々な操作をしていましたが、ここからはパソコンで作業します。

Messaging API チャンネルを作成するため、LINE Official Account Manager を開いてください。

- LINE Official Account Manager
 - <https://www.linebiz.com/jp/login/>

左側の「管理画面にログイン」から、LINE 公式アカウントの管理画面こと、LINE Official Account Manager にログインします。（図 2.4）



▲図 2.4 「管理画面にログイン」からログインする

LINE ビジネス ID^{*8}のログイン画面が表示されるので、[LINE アカウントでログイン]を選択します。(図 2.5)

[サービスに戻る](#)

LINE Business ID

LINEアカウントでログイン

または

ビジネスアカウントでログイン

[アカウントを作成](#)

LINEビジネスIDの利用については利用規約が適用されます。ログインすると規約に同意したものとみなされます。

[⑧ LINEビジネスIDについて](#)

日本語 ▾

[ヘルプ](#) [利用規約](#) © LINE Corporation

▲図 2.5 [LINE アカウントでログイン] を選ぶ

「1.3.1 LINE で LINE 公式アカウントを作ってみる」で登録したメールアドレスと、LINE のパスワード^{*9}を入力して、[ログイン] しましょう。入力が面倒な場合は、[QR コードログイン] から QR コードを表示して、それをスマートフォンの LINE の QR コードスキャンで読み込む形でもログイン可能です。(図 2.6)

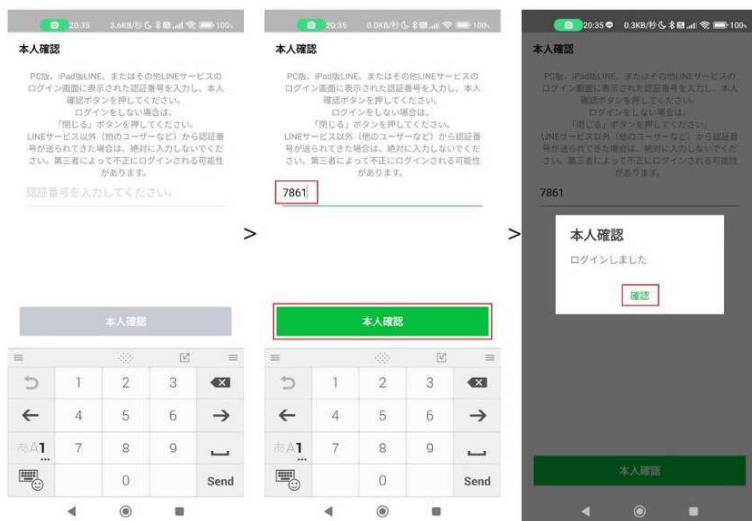
^{*8} LINE ビジネス ID は LINE Official Account Manager、LINE 公式アカウントの管理アプリ、LINE Developers コンソールなどで採用されている共通認証システムで、1 つのアカウントですべてに共通してログインできます。実は「1.4.1 管理アプリを使って LINE 公式アカウントからメッセージを送ってみる」で管理アプリにログインしたときも、この共通認証システムを使っていたのです。

^{*9} 「LINE のパスワードなんて設定したっけ？ 何も覚えていません」という人は LINE を開いて、下に並んでいる [ホーム] タブから右上の歯車アイコンをタップして [設定] を開き、[アカウント] の [パスワード] からすぐに新しいパスワードを設定できます。 <https://guide.line.me/ja/account-and-settings/account-and-profile/set-password.html>



▲図 2.6 「ログイン」すると 4 桁の認証番号が表示される

二要素認証のため、4 桁の認証番号が表示されます。スマートフォンの LINE を開くと、認証番号入力画面が表示されますので、この 4 桁の認証番号を入力して「本人確認」をタップします。「ログインしました」と表示されたら、「確認」をタップしてください。(図 2.7)



▲図 2.7 認証番号を入力して「本人確認」をタップする

これで LINE Official Account Manager にログインできました！（図 2.8）



▲図 2.8 LINE Official Account Manager にログインできた

2.3.2 Messaging API チャンネルを作って紐づける

LINE Official Account Manager にログインできたら、アカウントリストから、先ほど作った LINE 公式アカウントを選択します。（図 2.9）



▲図 2.9 先ほど作った LINE 公式アカウントを選択する

右上にある「設定」を開きます。(図 2.10)



▲ 図 2.10 右上にある「設定」を開く

左メニューの「設定」の配下にある「Messaging API」を開いて、「Messaging API を利用する」を押します。(図 2.11)



▲ 図 2.11 「Messaging API」を開いて「Messaging API を利用する」を押す

まだ LINE Developers コンソールにログインしたことがなかったため、開発者情報の入力を求められます。ここでいう「開発者」とは、イコール「LINE Developers コンソールにアクセスする人のこと」です。あなたの「名前」*¹⁰と「メールアドレス」*¹¹を入力して、リンク先の「LINE 開発者契約」を確認した上で、同意できる内容であれば「同意する」を押します。(図 2.12)

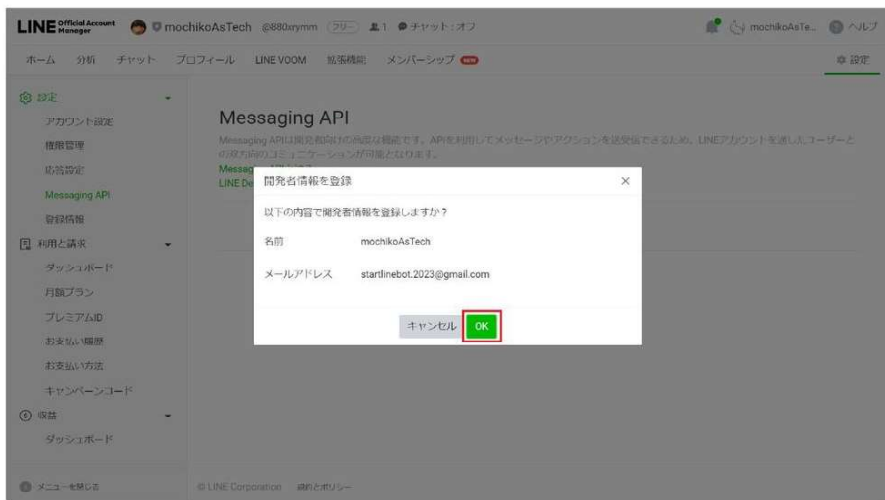


▲図 2.12 「名前」と「メールアドレス」を入力して「同意する」を押す

入力した開発者情報の確認画面が表示されます。表示されている内容で問題なければ、「OK」を押します。(図 2.13)

*¹⁰ 筆者は「名前」の欄に個人事業主としての屋号（mochikoAsTech）を記入していますが、個人開発者として登録するのであれば普通に氏名の入力で構わないと思います。

*¹¹ ここで登録するメールアドレスは、LINE に登録してあるメールアドレスとは別のアドレスでも構いません。開発者に対するお知らせを送ってほしいメールアドレスを記入しましょう。ただし LINE Developers コンソールへのログインに使用するメールアドレスは、LINE に登録してあるメールアドレスであって、ここで記入したメールアドレスではないので、その点は注意してください。

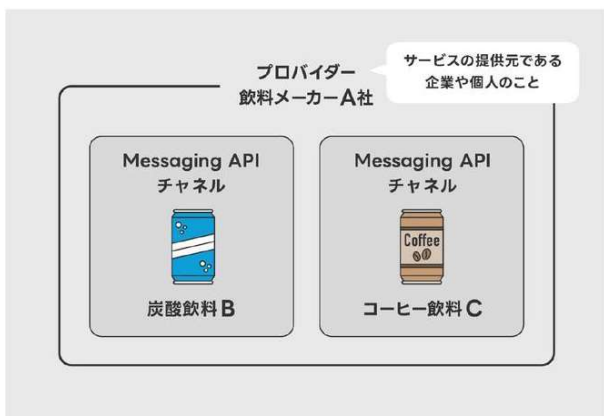


▲図 2.13 表示されている内容で問題なければ [OK] を押す

開発者情報を登録すると、今度はプロバイダーの選択画面が表示されます。プロバイダーとは、サービスを提供し、ユーザーの情報を取得する企業や開発者個人のことを指します。これから作成する Messaging API チャンネルは、このプロバイダーというものの下に属します。プロバイダーの下には、複数のチャンネルが所属できます。

運営元が個人だと「チャンネルも俺！ プロバイダーも俺！ 全部俺だ！」みたいな気持ちになって違いが分かりにくいですが、たとえば飲料メーカーの A 社が「炭酸飲料 B」と「コーヒー飲料 C」という 2 つのブランドを展開していた場合、プロバイダー名は「A 社」になり、その配下にある Messaging API チャンネル名は「炭酸飲料 B」や「コーヒー飲料 C」になります。^{*12} (図 2.14)

^{*12} 複数のチャンネルがあって、それらを 1 つのプロバイダー配下に収めたときと、それぞれプロバイダーを別にしたときで何が変わるかという、ユーザー ID の扱いが変わります。ユーザーを一意に識別するためのユーザー ID は、同じユーザーであってもプロバイダーごとに異なる値が発行されます。つまり mochiko さんというひとりのユーザーがいたとき、「A 社」というプロバイダーの配下にあるチャンネルから見た mochiko さんのユーザー ID と、また別の「B 社」というプロバイダーの配下にあるチャンネルから見た mochiko さんのユーザー ID は別々の値になるのです。そのため、本書では詳しく触れませんが LINE ログインチャンネルと Messaging API チャンネルでユーザー情報を連携したい場合は、その 2 つのチャンネルが同一のプロバイダー配下にいないければならない、などの制約があります。



▲図 2.14 チャンネルはプロバイダーの配下に属する

あなたが個人の開発者なのであれば、プロバイダー名は個人の名前やハンドルネームでも構いません。誰かに「この LINE 公式アカウントの運営元はどこなんですか?」と聞かれたときに、あなたが答えるであろう名称をプロバイダー名にしましょう。

まだプロバイダーというものを1つも持っていないので、今回は「プロバイダーを作成」します。プロバイダー名を入力して、「LINE 公式アカウント API 利用規約」を確認し、「同意する」を押します。(図 2.15)



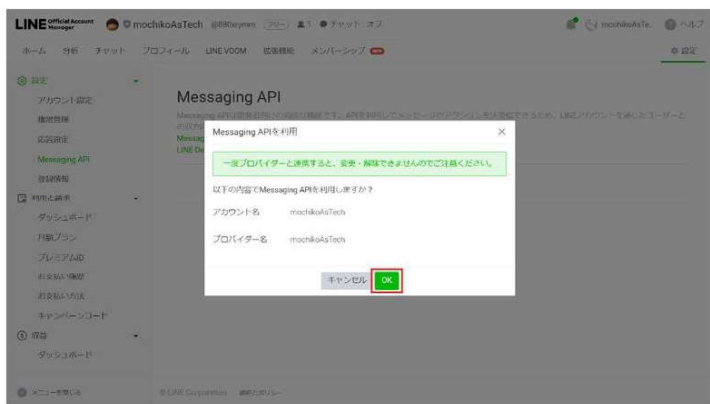
▲図 2.15 プロバイダー名を入力して「同意する」を押す

もしあなたがサービス提供者として、プライバシーポリシーや利用規約をすでに持っていたら、プロバイダーのプライバシーポリシーと利用規約としてここで URL を登録できます。個人開発者であればおそらくどちらも持っていないと思いますので、その場合は何も入力せずに [OK] を押して進んで構いません。(図 2.16)



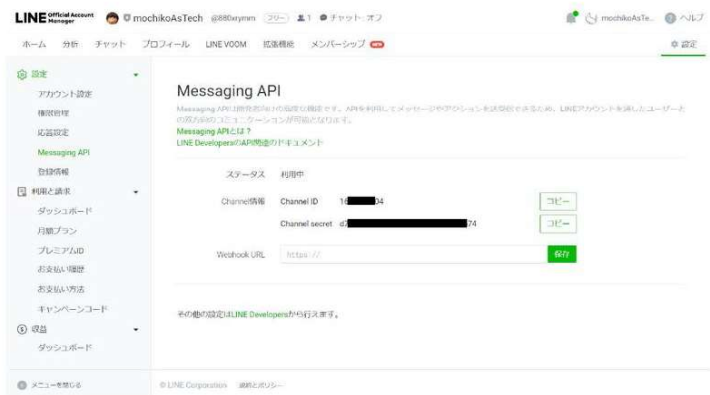
▲ 図 2.16 入力せずに [OK] を押す

この LINE 公式アカウントを、このプロバイダー配下の Messaging API チャンネルと紐づけますがいいですか、という最終確認の画面が表示されます。記載のとおり、一度チャンネルをプロバイダーと紐づけてしまうと、後から「やっぱりあっちのプロバイダー配下に移動させたい!」と思っても、絶対に移動できないので、プロバイダー名はよく確認してください。問題なければ [OK] を押します。(図 2.17)



▲図 2.17 確認して [OK] を押す

これで LINE 公式アカウントと紐づく Messaging API チャンネルができました！（図 2.18）



▲図 2.18 Messaging API チャンネルができた！

2.4 Messaging API を使ったメッセージ送信を試してみよう

それでは Messaging API チャンネルができたので、早速 Messaging API を使ってメッセージを送信してみましょう。

2.4.1 LINE Developers コンソールでチャンネルアクセストークンを発行する

Messaging API を使ってメッセージを送信するにあたって、前述したチャンネルアクセストークンというものが要るので、LINE Developers コンソールを開きます。

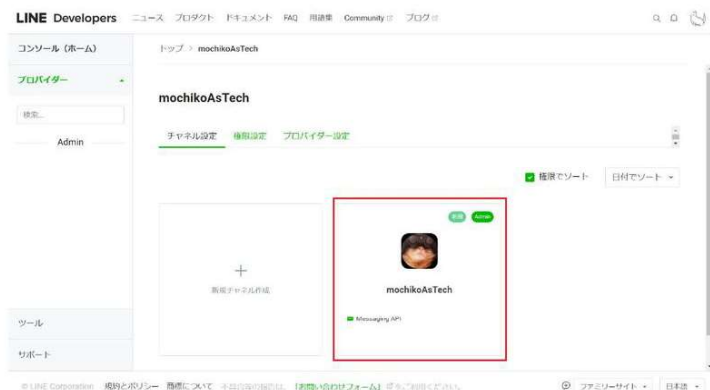
- LINE Developers コンソール
– <https://developers.line.biz/console/>

すでに LINE Official Account Manager にログインしていれば、上記の URL を開くと、そのまま LINE Developers コンソールのプロバイダー一覧が表示されるはずです。もしログインを求められたら、「2.3.1 LINE Official Account Manager にログインする」と同じように LINE のアカウントでログインしてください。プロバイダー一覧が表示されたら、左メニューで、先ほど作ったプロバイダーを選びます。(図 2.19)



▲図 2.19 さっき作ったプロバイダーを選ぶ

プロバイダーを選ぶと、そのプロバイダーの配下にあるチャンネルの一覧が表示されます。続いて、同じく先ほど作った Messaging API チャンネルを選びます。(図 2.20)



▲図 2.20 プロバイダー配下のチャンネル一覧から先ほどのチャンネルを選ぶ

Messaging API チャンネルが表示されたら、[チャンネル基本設定] タブの右隣にある [Messaging API 設定] タブを開いてください。(図 2.21)



▲図 2.21 先ほど作った Messaging API チャンネルが表示された

[Messaging API 設定] タブをいちばん下までスクロールすると、[チャンネルアクセストークン (長期)] があります。[発行] を押して、長期のチャンネルアクセストークンを発行してください。(図 2.22)



▲図 2.22 [発行] を押して長期のチャンネルアクセストークンを発行する

このチャンネルアクセストークン^{*13}は、Messaging API を使うときに、自分がその Messaging API チャンネルの持ち主であることを証明する身分証のような役割を果たします。うっかりチャンネルアクセストークンが載った画面をブログで公開したり、ソースコードに直接書いて GitHub に Push したりしないように注意してください。

長期のチャンネルアクセストークンを発行したら、右側のコピーボタンを押してコピーします。[コピーしました] と表示されます。(図 2.23)

^{*13} チャンネルアクセストークンにはいくつか種類がありますが、本書では説明を割愛し、この長期のチャンネルアクセストークンを使用します。 <https://developers.line.biz/ja/docs/messaging-api/channel-access-tokens/>



▲図 2.23 発行したチャネルアクセストークンをコピーする

いまコピーしたチャネルアクセストークンはこの後で何度も必要となります。忘れないように、パソコンのメモ帳にしっかりメモしておいてください。

続けて「チャネル基本設定」タブのチャネルシークレットもコピーして、一緒にメモしておきましょう。チャネルシークレットもこの後の手順で必要となります。(図 2.24)

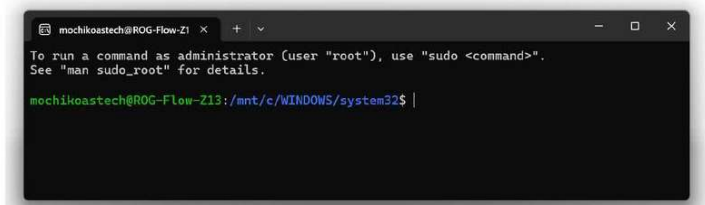


▲図 2.24 チャネルシークレットもコピーする

チャネルアクセストークンとチャネルシークレットが手に入ったので、早速 Messaging API でメッセージを送ってみましょう。

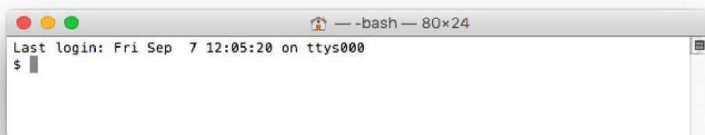
2.4.2 Messaging API でブロードキャストメッセージを送信する

実は、ただメッセージを送るだけならポットサーバーは必要ありません。あなたのパソコンで `curl` コマンドをたたくことで、Messaging API を使って LINE 公式アカウントからメッセージを送信できます。あなたが使っているパソコンが Windows なら WSL^{*14} (図 2.25) を起動してください。



▲図 2.25 Windows なら WSL を起動する

Mac を使っている方は、ターミナル (図 2.26) を起動してください。



▲図 2.26 Mac ならターミナルを起動する

WSL やターミナルがどこにあるのか分からないときは、Windows なら画面下部の検索ボックスで「WSL」と検索 (図 2.27)、Mac なら画面右上にある虫眼鏡のマークから Spotlight で「ターミナル」と検索 (図 2.28) すれば起動できます。

^{*14} WSL とは Windows Subsystem for Linux の略で、Windows 上で動く Linux 環境のことです。本書において WSL は「WSL2」というバージョンを想定しています。WSL2 は Windows 10 のバージョン 2004 (2020 年 5 月に配信されたパッチ) 以上、または Windows 11 であればデフォルトで動作します。Windows 10 なのに動かないよ！ という場合は、Windows Update をご確認ください。
<https://learn.microsoft.com/ja-jp/windows/wsl/install>



▲図 2.27 Windows なら検索ボックスで「WSL」と検索



▲図 2.28 Mac なら Spotlight で「ターミナル」と検索

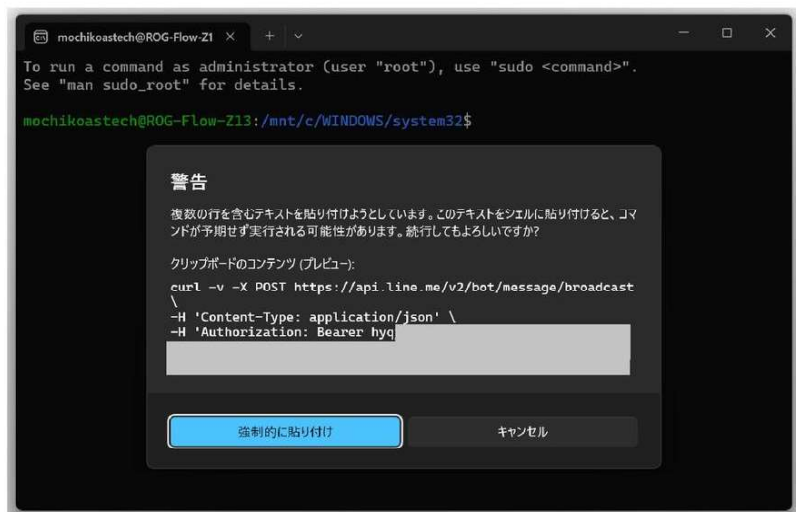
WSL またはターミナルが起動できたら、次の curl コマンド^{*15}（リスト 2.1）の 3 行目にある{チャンネルアクセストークン}の部分、Messaging API チャンネルのチャンネルアクセストークンに置き換えてください。チャンネルアクセストークンは、先ほど「2.4.1 LINE Developers コンソールでチャンネルアクセストークンを発行する」でコピーしましたね。

▼リスト 2.1 curl コマンドで友だちにメッセージを送る

```
1: curl -v -X POST https://api.line.me/v2/bot/message/broadcast \
2: -H 'Content-Type: application/json' \
3: -H 'Authorization: Bearer {チャンネルアクセストークン}' \
4: -d '{
5:     "messages": [
6:         {
7:             "type": "text",
8:             "text": "Messaging APIでメッセージを送信しています。"
9:         },
10:        {
11:            "type": "sticker",
12:            "packageId": "6325",
13:            "stickerId": "10979905"
14:        }
15:    ]
16: }'
```

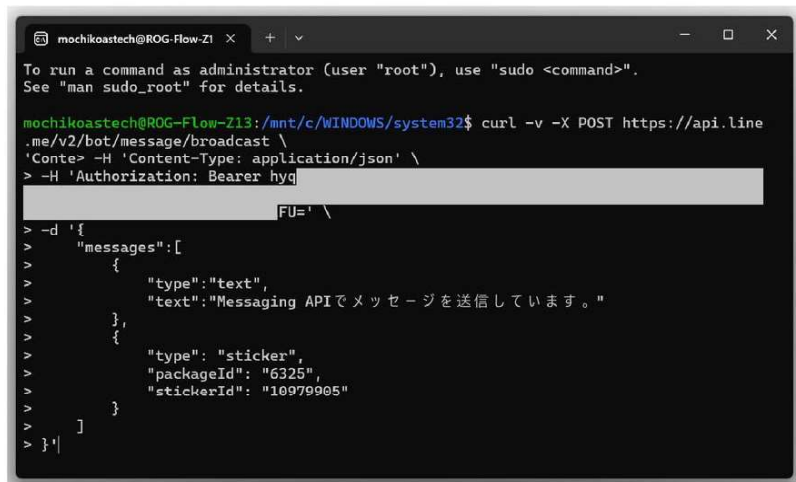
チャンネルアクセストークンを置き換えたら、curl コマンドをまるごとコピーして WSL もしくはターミナルに貼り付けます。WSL の場合は、複数行をまとめて貼り付けると警告が出ますが、[強制的に貼り付け] を押します。（図 2.29）

^{*15} このコードをはじめとして、本書に出てくるコードはいずれも GitHub にある本書のリポジトリで公開されています。PDF から直接コピー＆ペーストすると行番号などが混入してしまうため、ぜひこちらを使ってください。 <https://github.com/mochikoAsTech/startLINEBot/blob/master/articles/send-broadcast-message.sh>



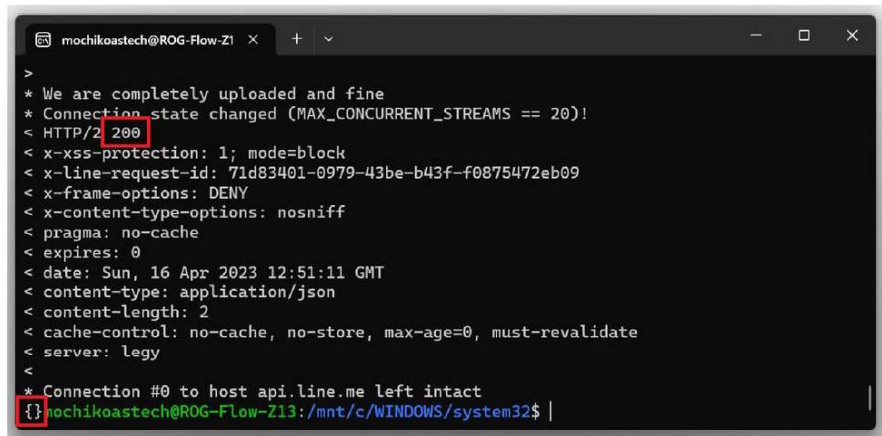
▲ 図 2.29 複数行の貼り付けに対する警告が出たら [強制的に貼り付け] を押す

貼り付けたら Enter を押して curl コマンドを実行します。(図 2.30)



▲ 図 2.30 貼り付けたら Enter を押して実行する

レスポンスが画面に出力されます。curl コマンドを使って API をたたいた結果、レスポンスとしてステータスコード 200^{*16}と空の JSON が返ってきたことが分かります。(図 2.31)



```

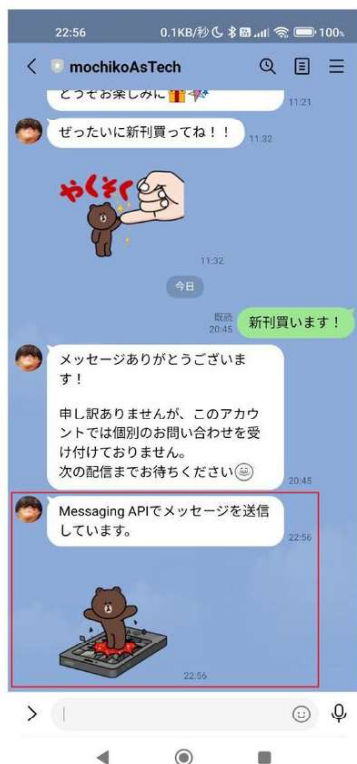
>
* We are completely uploaded and fine
* Connection state changed (MAX_CONCURRENT_STREAMS == 20)!
< HTTP/2 200
< x-xss-protection: 1; mode=block
< x-line-request-id: 71d83401-0979-43be-b43f-f0875472eb09
< x-frame-options: DENY
< x-content-type-options: nosniff
< pragma: no-cache
< expires: 0
< date: Sun, 16 Apr 2023 12:51:11 GMT
< content-type: application/json
< content-length: 2
< cache-control: no-cache, no-store, max-age=0, must-revalidate
< server: legy
<
* Connection #0 to host api.line.me left intact
{} mochikoastech@ROG-Flow-Z13:/mnt/c/WINDOWS/system32$

```

▲図 2.31 ステータスコード 200 が返ってきた

すると、Messaging API で送ったメッセージが LINE に届きました！(図 2.32)

^{*16} ステータスコードとは、サーバーに対して「ウェブページを見せてくれ！」とか「天気情報をくれ！」といったリクエストを投げた際に、サーバーから返ってくる 3 桁の数字で、リクエストの結果を表すものです。たとえば 200 OK ならリクエストが成功したこと、404 Not Found ならリクエストされたリソース（ページなど）が見つからなかったこと、503 Service Unavailable なら過負荷などによってリクエストを処理できなかったことをそれぞれ表しています。ちなみにステータスコード（3 桁の数字）の後ろに付いている OK や Not Found のようなテキストは **reason-phrase** と言って、ステータスコードの意味を人間が理解しやすいよう簡潔に説明しているものです。



▲図 2.32 Messaging API で送ったメッセージが届いた

やったあ！ Messaging API をたたいてメッセージが送れましたね！ おめでとうございます。

いまあなたがたたいたのは、LINE 公式アカウントと友だちになっている人全員にメッセージを一齐送信するための「ブロードキャストメッセージを送る」という API です。もし curl コマンドを実行した後にステータスコード 200 以外が返ってきて、LINE にメッセージが届かなかった場合は、公式ドキュメントの API リファレンス^{*17}でこの「ブロードキャストメッセージを送る」という API のエラーレスポンス例を確認してみてください。

^{*17} ブロードキャストメッセージを送る | Messaging API リファレンス | LINE Developers
<https://developers.line.biz/ja/reference/messaging-api/#send-broadcast-message>

メッセージが届いたら、curl コマンドの text の部分を好きなテキストにしたり、公開されている「送信可能なスタンプリスト*18」を見ながら curl コマンドの packageId や stickerId を好きなものに変えてみたりして、メッセージを何度か送り直してみましょう。(図 2.33)



▲図 2.33 テキストやスタンプを変えてメッセージを何度か送ってみよう

curl コマンドで API をたたくと、手元のスマートフォンで LINE にメッセージが届く。こうして自分がやったことが、ちゃんと動いて、結果が目に見えるのはとってもうれしいですね。

*18 送信可能なスタンプリスト | LINE Developers

<https://developers.line.biz/ja/docs/messaging-api/sticker-list/>

【コラム】仕事で LINE Bot を開発するときにも「個人の LINE アカウント」が必要か？

Messaging API を使って開発をするとき、さまざまな設定をするための管理画面が「LINE Developers コンソール」です。この LINE Developers コンソールにログインするには、LINE アカウントまたはビジネスアカウントのどちらかが必要です。

すでにスマートフォンで LINE を使っていれば、その LINE アカウントでそのままログインできます。本書では、第1章「LINE 公式アカウントをつくってみよう」で LINE 公式アカウントを作ったときの LINE アカウントで、LINE Developers コンソールにログインしています。

個人開発であればこれで問題ありませんが、仕事の場合は「私物のスマホに入れている個人の LINE アカウントを業務で使うのはちょっと…」というケースももちろんあると思います。その場合は業務用のメールアドレスでビジネスアカウントを作りましょう。ただしビジネスアカウントで LINE Developers コンソールにログインした場合は、Messaging API チャンネルが作れないなど、できることに一部制限^{*19}があります。

その場合は、プロジェクトの中で誰かひとりが会社の検証端末に LINE を入れ、その LINE アカウントで LINE Developers コンソールにログインして、Messaging API チャンネルを作ればいいのです。後はそのプロバイダーやチャンネルに、他のメンバーのビジネスアカウントを Admin 権限で追加してやれば、開発を進める上で問題はないでしょう。

2.5 LINE 公式アカウントから友だちに返信するには

さて、ここまでは「LINE 公式アカウントからメッセージを送る」という、LINE 公式アカウント起点の話をしてきました。LINE 公式アカウント起点というのは、たとえば月初に今月の定休日を知らせるメッセージを送るとか、新商品が出たときにメッセージを送るとかいうように、LINE 公式アカウント側が「送りたい！」と思った任意のタイミングでメッセージを送る、ということです。

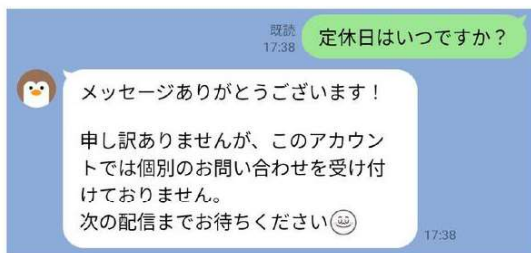
^{*19} LINE Developers コンソールへのログイン | LINE Developers <https://developers.line.biz/ja/docs/line-developers-console/login-account/>

でも片方が話したいときだけ一方的に話すのでは、それは対話とは言えません。ユーザーが友だち追加してくれたときや、ユーザーが何か質問してきたとき、ユーザーがリッチメニュー*20をタップしたときなど、ユーザー起点のコミュニケーションにきちんと反応できるようになってこそそのチャットボットです。

ではユーザー起点のメッセージに LINE 公式アカウントが反応するには、どうしたらいいのでしょうか？ ユーザーからのメッセージに対応する方法は、大きく分けて4つあります。

2.5.1 方法 1. 固定の自動応答を設定しておいて個別対応は一切しない

1 つめは、LINE Official Account Manager で次のような固定の「応答メッセージ」を設定しておいて、個別の対応は一切しないという方法です。「1.4.4 LINE 公式アカウントにメッセージを送ってみる」で「新刊買います！」というメッセージを送ったときに受け取った返信なので見覚えがありますね。(図 2.34)



▲図 2.34 自動の応答メッセージ

この自動の応答メッセージは、LINE 公式アカウントを作ったときにデフォルトで設定されているので、あなたも友だち追加した LINE 公式アカウントでよく似たメッセージを見たことがあるかもしれません。

これだと個別対応のコストが一切要らなくなるものの、ユーザー起点のコミュニケーションはできなくなるので、LINE 公式アカウントとのトークは一方的な宣伝をするだけの場所になります。ただユーザーにとっては、話しかけたのに何も応答してもらえず無視されるよりは、このようなメッセージが返ってきた方がまだ印象が良いかもしれません。

*20 リッチメニューについては、「3.2 リッチメニュー」で後述します。

2.5.2 方法 2. 人間が手打ちのチャットで返信する

2 つめは、「中の人」が頑張って手打ちのチャットで返信する、という方法です。LINE Official Account Manager や管理アプリにはチャットの機能があるので、「中の人」がその機能でユーザーからのメッセージを確認して、手打ちのチャットで頑張って対応します。

たとえば個人で経営する小さなヘアサロンで、予約を電話ではなく LINE のトークで受け付けたい、というような場合は、このチャットで必要十分なケースもあるでしょう。

2.5.3 方法 3. 内容に応じた自動応答メッセージで対応する

3 つめは、内容に応じた自動応答メッセージを用意しておいて自動で返信する、という方法です。

LINE Official Account Manager や管理アプリには、応答メッセージと AI 応答メッセージ^{*21}というものがあります。応答メッセージはキーワードとメッセージを事前に設定しておくことで、ユーザーがそのキーワードと完全一致するメッセージを送ってきたら、自動でメッセージを返信する機能です。完全に一致する必要があるので、たとえば「メニュー」というキーワードの場合、ユーザーが「メニュー」とだけ送ってくれば応答できますが、「メニューは？」と送ってきた場合は応答できません。

一方、AI 応答メッセージはもう少し融通が利いて、ユーザーから送られたメッセージの内容を AI が判別して、適切なメッセージで自動で返信してくれます。たとえば飲食店なら、営業時間、住所、Wi-Fi、コンセント、駐車場、喫煙可否など、質問が想定されることについてそれぞれ事前に返信メッセージを設定しておくことで、単語が完全に一致しなくてもよしなに回答してくれます。たとえば「Wi-Fi」に対して「申し訳ありませんが Wi-Fi はご利用いただけません」という回答をオンにしておくと、「Wi-Fi ってありますか?」のようなメッセージだけに限らず、「ネットって使えたりする?」「無線ってありますか?」「無料のわいふあいて使える?」といったメッセージでもきちんとその回答が返ってきます。かしこい…! 開発不要で使える AI チャットボットとも言えますね。

応答メッセージと AI 応答メッセージは、どちらか片方だけ使うこともできますし、キーワードで完全一致したら応答メッセージを返して、そうでない場合は AI 応答メッセージを返す、というように両方使うこともできます。

さらに方法 2 と方法 3 を併用することも可能です。LINE Official Account Manager

^{*21} 自動応答が可能に! 「応答メッセージ」と「AI 応答メッセージ」とは | LINE for Business <https://www.linebiz.com/jp/column/technique/20191128/>

や管理アプリで営業時間を設定しておくことで、営業時間内は中の人が手打ちのチャットで応答して、営業時間外は自動応答のメッセージに任せる、というような運用ができます。

2.5.4 方法 4. Messaging API で返信する

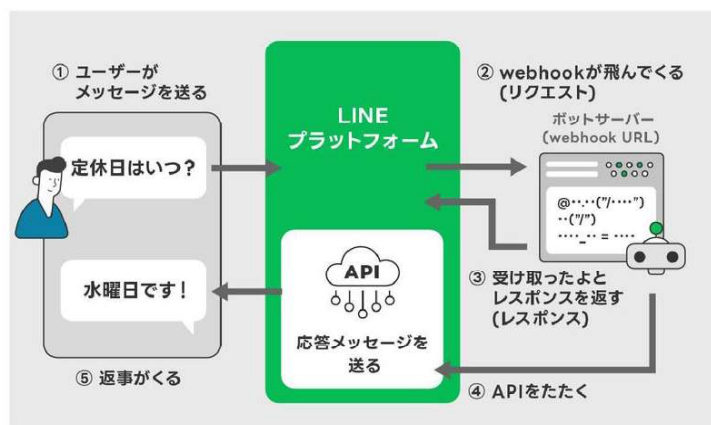
4 つめは Messaging API で返信する、という方法です。Messaging API では、この「ユーザー起点のコミュニケーション」に気づく方法として、Webhook というものが用意されており、この Webhook をボットサーバー^{*22}で受け止めることでボットから返信ができます。では Webhook とはいったいどんなものなのでしょう？

2.5.5 Webhook とは

ユーザーが、LINE 公式アカウントを友だち追加したり、ブロックしたり、LINE 公式アカウントにメッセージを送ったりというように、何かしらの「イベント」を起こすと、LINE プラットフォームからボットサーバーに対して、Webhook が送られます^{*23}。LINE プラットフォームから送られてくる Webhook を受け止めるために、LINE 公式アカウントを運営するあなたは、自分でこのボットサーバーを用意する必要があります。(図 2.35)

^{*22} ボット（自動応答するためのプログラム）が稼働しているウェブサーバーのことをボットサーバーと呼びます。

^{*23} 実際に飛んでくる Webhook がどんなものなのかは、「2.6.5 LINE プラットフォームから飛んできた Webhook を目視確認する」で見られます。



▲図 2.35 ユーザーがメッセージを送るとボットサーバーに Webhook が飛んでくる

通常、私たちはブラウザでショッピングサイトのマイページを開いて、住所を入力して送信ボタンを押すことで、ウェブサーバーに「住所情報をこれに変更してくれ！」とリクエストを投げ、リクエストを受けたウェブサーバーが諸々の処理をしてから「はい。住所変更終わりましたよ」と変更完了ページをレスポンスで返してくれたりします。このときは、ウェブサーバー側が「サーバー」で、自分やブラウザ側が「クライアント」です。

しかし Webhook においてはこれが逆転します。LINE プラットフォームから「ユーザーがこんなメッセージを送ってきたよ！ Webhook を受け取って！」とリクエストが飛んでくるので、あなたは用意したボットサーバーでそれを受け取って、「ありがとう！ Webhook ちゃんと受け取ったよ！ ステータスコード 200！」というレスポンスを返さなければいけません。Webhook を投げってくる LINE プラットフォームが「クライアント」で、あなたの用意するボットサーバーが「サーバー」なのです。

私たちはクライアントの立場は頻繁に経験していますが、サーバーの立場になる経験はあまりないのでなんだか不思議な感じがしますね。自分が「ボットサーバーを用意する側」であり、「Webhook を受け取ってレスポンスを返す側」だ、と言われてもまだあんまりピンとこないかもしれません。この説明だけで今すぐに Webhook を完璧に理解できなくても大丈夫です。手を動かしてボットサーバーを用意し、実際に Webhook を受け取ってみましょう。

【コラム】LINE 公式アカウントの「既読」はいつ付くのか？

通常、友だちに送った LINE のメッセージには、相手がトークルームを開いてメッセージを見ることで「既読」がつきます。では LINE 公式アカウントの場合は、「既読」はいつ付くのでしょうか？

LINE Official Account Manager や管理アプリの応答設定に「チャット」と「Webhook」という設定項目があります。この「チャット」をオンにしていると、中の人（チャット画面）でメッセージを確認するまでは、LINE 公式アカウント側でメッセージを読んだことを示す「既読」マークがつきません。逆にこの「チャット」をオフにしていると、メッセージが送られた瞬間に自動的に「既読」がつきます。

Messaging API を使って LINE 公式アカウントから自動応答しているのに、なぜか「既読」マークが付かない！というときは、うっかり応答設定で「チャット」をオンにしないか確認してみてください。応答設定で「Webhook だけをオンにしている」と自動で既読が付きますが、「チャットだけをオンにしている」もしくは「チャットと Webhook の両方をオンにしている」状態だと、チャットの画面でメッセージを確認するまで既読が付きません。

「チャット」も「Webhook」も両方オンにして併用したいけど、チャット画面で確認したときではなく自動で既読を付けたい、という場合は「既読 API^{*24}」という API を使うことで任意のタイミングで既読が付けられます。ただ残念ながら既読 API は法人ユーザー限定のオプション機能です。

2.6 オウム返しするチャットボットを作ってみよう

メッセージを Webhook で受け取って、ボットから自動返信する一連の流れを理解するため、まずは一番簡単な「オウム返しするチャットボット」を作ってみましょう。

^{*24} 既読 API | LINE Developers

<https://developers.line.biz/ja/docs/partner-docs/mark-as-read/>

2.6.1 Messaging API の SDK を準備する

Messaging API では、開発をサポートする SDK が Java、PHP、Go、Perl、Ruby、Python、Node.js で用意^{*25}されています。SDK は Software Development Kit の略で、名前のとおり開発に必要なライブラリが詰まった工作キットのようなものです。たとえば料理でも、材料を揃えて野菜の皮を剥くところからやるととても大変ですが、下ごしらえの済んだ材料とレシピがひとまとめになっているミールキットを使えば、誰でも短時間で失敗もなく美味しい食事が作れます。それと同じで、開発においても SDK が用意されていたら、その SDK を使うことで色々ラクができます。

今回は Python という言語の SDK を使ってコードを書いていきます。

- LINE Messaging API SDK for Python
 - <https://github.com/line/line-bot-sdk-python>

それでは SDK を使うための準備をしていきましょう。Windows と Mac で手順が異なるので、それぞれ順番に説明していきます。

Windows の場合

「2.4.2 Messaging API でブロードキャストメッセージを送信する」で使用した WSL を再び起動して、次のコマンドを順番にたたいていきます。\$ は WSL のプロンプトを表していますので入力しないでください。

まずは `cd` コマンドでホームディレクトリ^{*26}に移動して、`mkdir` コマンドで `python`^{*27} というディレクトリを作ります。ls コマンドで確認して「python」と表示されたら、問題なく `python` ディレクトリが作成できていますので、作成した `python` ディレクトリの中に `cd` コマンドで移動してください。(図 2.36)

```
$ cd
$ mkdir python
$ ls
```

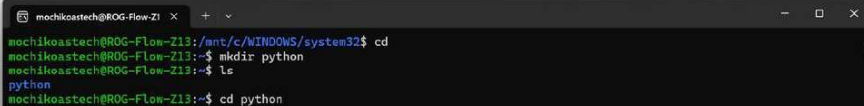
^{*25} LINE Messaging API SDK | LINE Developers

<https://developers.line.biz/ja/docs/messaging-api/line-bot-sdk/>

^{*26} 前述のとおり WSL は Windows 上で動く Linux 環境であり、Linux ではフォルダのことをディレクトリと呼びます。なので、ここではディレクトリと書いてあったら「ああ、フォルダのことだな」と思ってください。

^{*27} この作成するディレクトリの名前は必ず `python` にしてください。ディレクトリ名を `python` 以外にするとこの後の手順で正常に動きません。

```
$ cd python
```



```
mochikoastech@ROG-Flow-Z1 X + v
mochikoastech@ROG-Flow-Z13:/mnt/c/WINDOWS/system32$ cd
mochikoastech@ROG-Flow-Z13:~$ mkdir python
mochikoastech@ROG-Flow-Z13:~$ ls
python
mochikoastech@ROG-Flow-Z13:~$ cd python
```

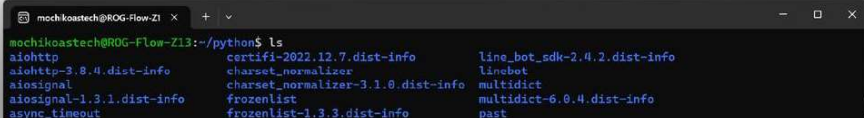
▲図 2.36 python ディレクトリを作ってその中に移動する

続いて、SDK をパソコンの中に取りってくるために pip コマンドが使いたいのので、apt コマンドで pip コマンドを連れてきます。sudo apt update をたたくとパスワードを聞かれるので、パソコンを起動したときに入力するのと同じパスワードを入力して Enter を押してください。sudo apt install python3-pip をたたくと、メッセージがだだーっと流れた後に Do you want to continue? [Y/n] と聞かれるので、Y を入力して Enter を押します。

```
$ sudo apt update
$ sudo apt install python3-pip
```

いよいよ pip コマンドで SDK をパソコンの中を取得してきます。SDK を取ってきたら、ls コマンドで python ディレクトリの中身を確認してみましょう。中身がなんだかたくさん入っていれば成功です。(図 2.37)

```
$ pip install line-bot-sdk -t . --no-user
$ ls
```



```
mochikoastech@ROG-Flow-Z1 X + v
mochikoastech@ROG-Flow-Z13:~/python$ ls
aiobot      certifi-2022.12.7.dist-info  line_bot_sdk-2.4.2.dist-info
aiosignal   charset-normalizer          linebot
aiosignal-1.3.1.dist-info  charset-normalizer-3.1.0.dist-info  multidict
async_timeout  frozenlist                  multidict-6.0.4.dist-info
                frozenlist-1.3.3.dist-info          past
```

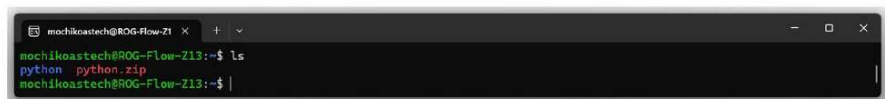
▲図 2.37 取ってきた SDK が python ディレクトリの中にみっちり入っている

取ってきた SDK をぎゅっと圧縮して ZIP に固めたいので、`cd` コマンドで1つの上のディレクトリに移動しましょう。そして `apt` コマンドで `zip` コマンドを連れてきます。`sudo apt install zip` をたたくと、メッセージがだだーっと流れた後に `Do you want to continue? [Y/n]` と聞かれるので、`Y` を入力して `Enter` を押してください。

```
$ cd ..  
$ sudo apt install zip
```

それでは `zip` コマンドで `python` ディレクトリをぎゅっと ZIP に固めましょう。`ls` コマンドで `python.zip` と `python` ディレクトリが確認できたら、これで SDK は準備完了です。(図 2.38)

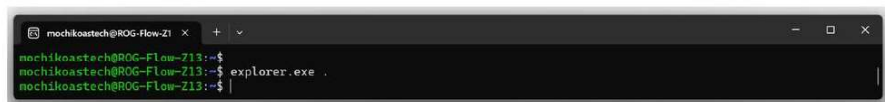
```
$ zip -r python.zip python  
$ ls
```



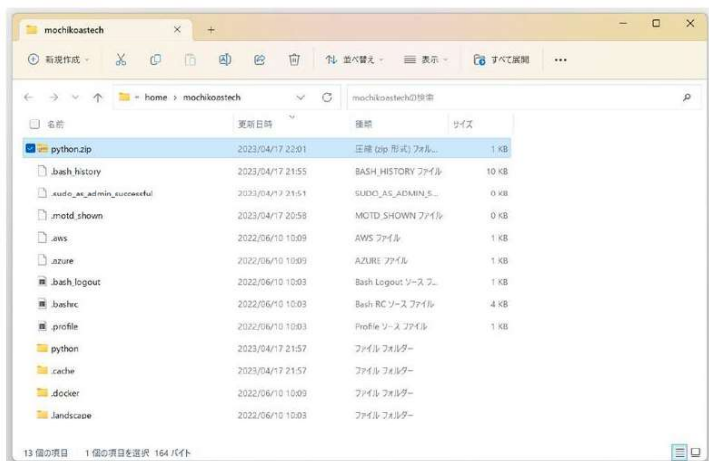
▲図 2.38 `zip` コマンドで `python` ディレクトリを ZIP に固めて `ls` コマンドで確認

最後に `explorer.exe .` をたたくと、WSL で見ていたディレクトリがエクスプローラで表示されます。(図 2.39、図 2.40)

```
$ explorer.exe .
```

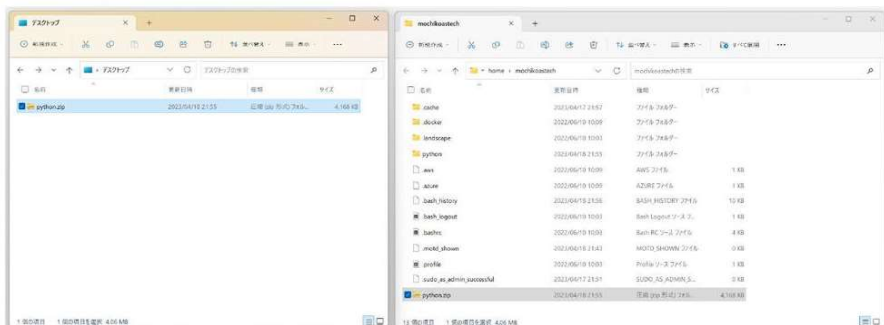


▲図 2.39 「`explorer.exe .`」をたたく



▲図 2.40 するとエクスプローラで python.zip のあるフォルダが表示される

作成した python.zip はこの後すぐに使うので、デスクトップにコピーしておきましょう。(図 2.41)



▲図 2.41 python.zip はデスクトップにコピーしておく

これで Messaging API の SDK が準備できました。「2.6.2 AWS Lambda と API Gateway でボットサーバーを作る」に進んでください。

Mac の場合

必要なコマンド*28を使えるようにするため、まずは Python の公式サイトから Python のインストーラーをダウンロードしてきます。[Download the latest version for macOS] の下にある [Download Python 3.x.x] をクリック*29してください。(図 2.42)

- Download Python | Python.org
 - <https://www.python.org/downloads/>



The screenshot shows the Python.org homepage. At the top, there's a navigation bar with links for Python, PEP, Docs, PyPI, Jobs, and Community. Below this is a search bar and a 'Donate' button. The main content area features a large banner with the text 'Download the latest version for macOS' and a button labeled 'Download Python 3.11.3'. Below the button, there are links for 'Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other' and 'Want to help test development versions of Python? Prereleases, Docker images'. To the right of the banner is an illustration of two parachutes carrying boxes. Below the banner, there's a section titled 'Active Python Releases' with a link to 'For more information visit the Python Developer's Guide.' and a table of active releases.

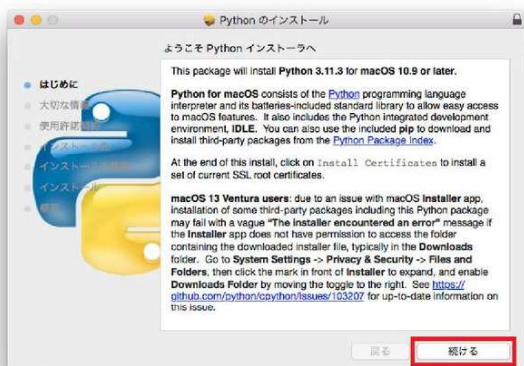
Python version	Maintenance status	First released	End of support	Release schedule
3.11	bugfix	2022-10-24	2027-10	PEP 664
3.10	bugfix	2021-10-04	2026-10	PEP 613
3.9	maintenance	2020-10-05	2025-10	PEP 622

▲ 図 2.42 Python のインストーラーをダウンロードする

*28 あなたの Mac ですでに pip3 コマンドが使える状態であれば、この Python をインストールする手順は飛ばして構いません。

*29 画像は 3.11.3 ですが、Python3 系でこれよりも新しいバージョンが出ていたら、そちらをダウンロードしてください。

ダウンロードした [python-3.x.x-macos11.pkg] を開きます。インストーラの指示に従って Python をインストールしてください。(図 2.43)



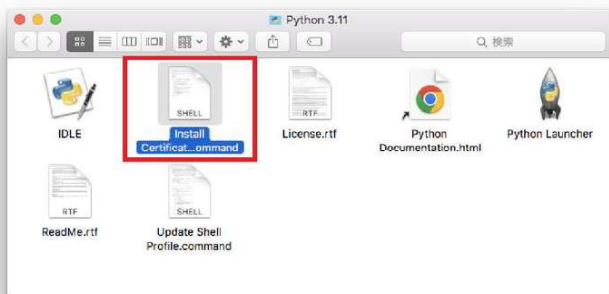
▲図 2.43 インストーラの指示に従って Python をインストールする

Python のインストールが完了したら [閉じる] をクリックします。(図 2.44)



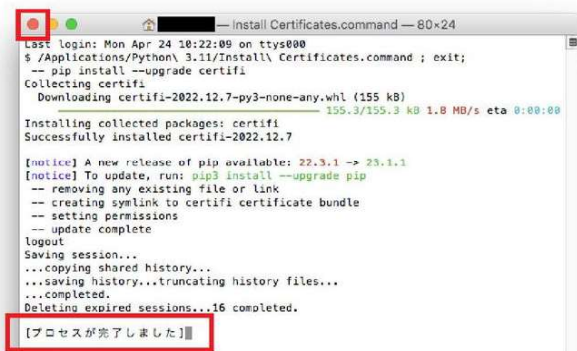
▲図 2.44 インストールが完了したら [閉じる] をクリックする

インストール完了後、自動的に表示される Finder で [Install Certificates.command] をダブルクリックして開きます。(図 2.45)



▲図 2.45 [Install Certificates.command] を開く

ターミナルが自動で起動して証明書のインストールが行われます。[プロセスが完了しました] と表示されたら、そのターミナルは閉じて構いません。(図 2.46)



▲図 2.46 [プロセスが完了しました] と表示されたらそのターミナルは閉じる

これで必要なコマンドが使えるようになったはずです。

それでは「2.4.2 Messaging API でブロードキャストメッセージを送信する」で使用したターミナルを再び起動して、次のコマンドを順番にたたいていきます。`$`はターミナルのプロンプトを表していますので入力しないでください。

`mkdir` コマンドで `python`^{*30} というディレクトリを作ります。`ls` コマンドで確認して「`python`」と表示されたら、問題なく `python` ディレクトリが作成できていますので、作成した `python` ディレクトリの中に `cd` コマンドで移動してください。(図 2.47)

```
$ mkdir python
$ ls -d python
$ cd python
```

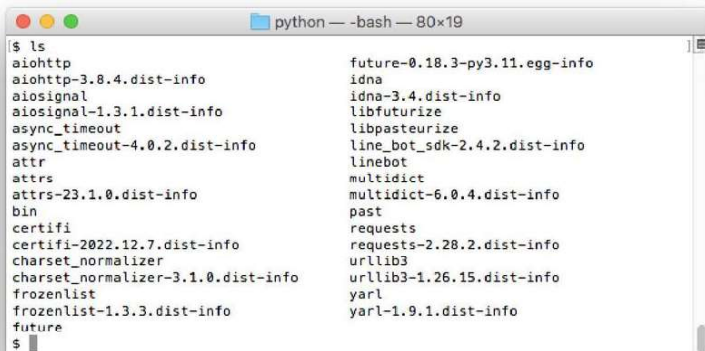


▲図 2.47 `python` ディレクトリを作ってその中に移動する

`pip3` コマンドで SDK をパソコンの中を取得してきます。SDK を取ってきたら、`ls` コマンドで `python` ディレクトリの中身を確認してみましょう。中身がなんだかたくさん入っていれば成功です。(図 2.48)

^{*30} この作成するディレクトリの名前は必ず `python` にしてください。ディレクトリ名を `python` 以外にするとこの後の手順で正常に動きません。

```
$ pip3 install line-bot-sdk -t . --no-user
$ ls
```

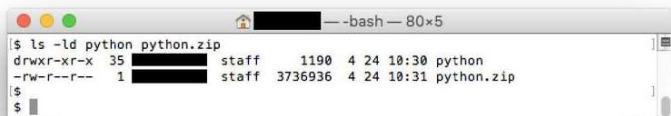


```
python -- bash -- 80x19
$ ls
aiohttp                                future-0.18.3-py3.11.egg-info
aiohttp-3.8.4.dist-info                 idna
aiosignal                              idna-3.4.dist-info
aiosignal-1.3.1.dist-info               libfuturize
async_timeout                          libpasteurize
async_timeout-4.0.2.dist-info           line_bot_sdk-2.4.2.dist-info
attr                                    linebot
attrs                                  multidict
attrs-23.1.0.dist-info                 multidict-6.0.4.dist-info
bin                                    past
certifi                                requests
certifi-2022.12.7.dist-info             requests-2.28.2.dist-info
charset_normalizer                     urllib3
charset_normalizer-3.1.0.dist-info      urllib3-1.26.15.dist-info
frozenlist                             yarl
frozenlist-1.3.3.dist-info             yarl-1.9.1.dist-info
future
$
```

▲図 2.48 取ってきた SDK が python ディレクトリの中にみっちり入っている

取ってきた SDK を圧縮して ZIP に固めたいので、cd コマンドで 1 つの上のディレクトリに移動しましょう。zip コマンドで python ディレクトリをぎゅっと ZIP に固めます。ls コマンドで python.zip と python ディレクトリが確認できたら、これで SDK は準備完了です。(図 2.49)

```
$ cd ..
$ zip -r python.zip python
$ ls -ld python python.zip
```



```
$ ls -ld python python.zip
drwxr-xr-x  35 [redacted] staff   1190  4 24 10:30 python
-rw-r--r--   1 [redacted] staff 3736936  4 24 10:31 python.zip
$
```

▲図 2.49 zip コマンドで python ディレクトリを ZIP に固めて ls コマンドで確認

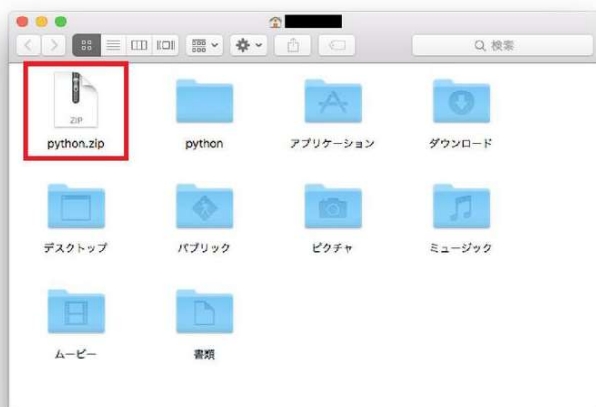
最後に **open .** をたたくと、ターミナルで見ていたディレクトリが Finder で表示されます。

```
open .
```



```
$ open .
```

▲図 2.50 「open .」をたたく



▲図 2.51 すると Finder で python.zip のあるフォルダが表示される

作成した `python.zip` はこの後すぐに使うので、デスクトップにコピーしておきましょう。

これで Messaging API の SDK が準備できました。

2.6.2 AWS Lambda と API Gateway でボットサーバーを作る

今回は Webhook を受け取ってレスポンスを返すボットサーバーとして、AWS のサーバーレスサービス、AWS Lambda と API Gateway を使用します。

AWS を初めて使用する場合、AWS アカウントを作成してから 1 年間は利用料が無料となります。ただし無料利用枠の範囲は決まっており、何をどれだけ使っても無料という訳ではありません。どのサービスをどれくらい無料で使えるのか？ は「AWS 無料利用枠」のページに記載されていますので、そちらを参照してください。AWS アカウントを持っていない場合は、同じ「AWS 無料利用枠」のページにある「無料アカウントの作成」から作成してください。^{*31}（図 2.52）

^{*31} なお AWS アカウントの作成の詳しい手順は「DNS をはじめよう」という書籍で、また AWS とは何かについての説明や、無料利用枠の範囲、利用金額が一定額を超えたらアラートが飛ぶようにする設定などは「AWS をはじめよう」という書籍で詳しく紹介しています。もし AWS アカウントの作成や設定に不安がある場合は、そちらを参考にしてください。 <https://mochikoastech.booth.pm/>

- AWS 無料利用枠
 - <https://aws.amazon.com/jp/free/>



▲図 2.52 AWS 無料利用枠

それでは早速、AWS のマネジメントコンソールを開いて、上部の検索窓で **lambda** と検索し、AWS Lambda を開きます。(図 2.53)

- AWS マネジメントコンソール
 - <https://console.aws.amazon.com/>



▲図 2.53 検索窓から Lambda を開く

Messaging API SDK のレイヤーを作成する

AWS Lambda を開いたら、左メニューの「レイヤー」から、「レイヤーの作成」をクリックします。(図 2.54)



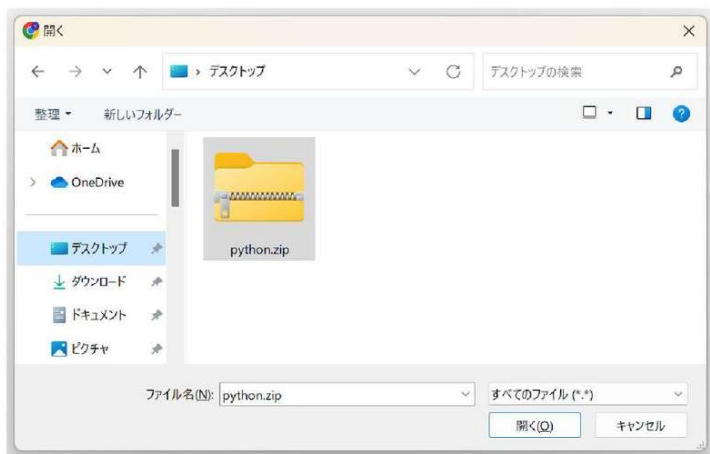
▲図 2.54 「レイヤーの作成」をクリック

レイヤー設定の「名前」や「説明」などは次のように入力します。(表 2.1)

▼表 2.1 レイヤー設定

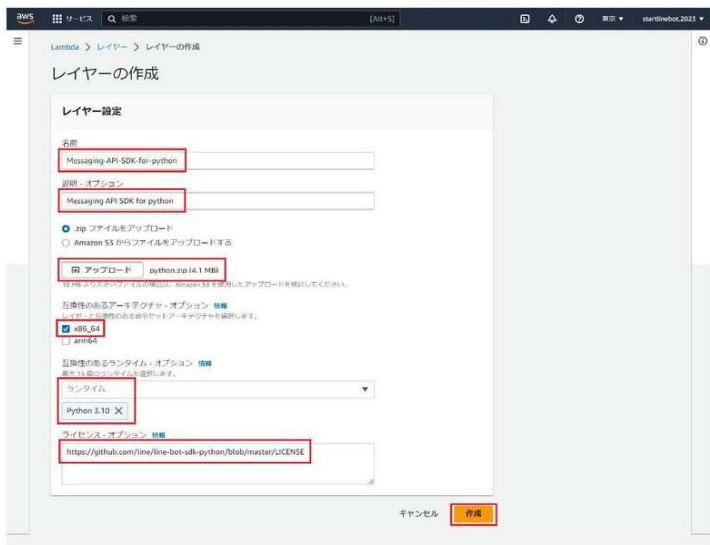
名前	Messaging-API-SDK-for-python
説明	Messaging API SDK for python
アップロード方法	zip ファイルをアップロード
アップロードするファイル	「2.6.1 Messaging API の SDK を準備する」で用意した python.zip
互換性のあるアーキテクチャ	x86_64 にチェックを入れる
互換性のあるランタイム	Python 3.10 を選択
ライセンス	https://github.com/line/line-bot-sdk-python/blob/master/LICENSE

「アップロード」をクリックしたら、「2.6.1 Messaging API の SDK を準備する」で準備しておいた、デスクトップの python.zip を選択し、アップロードしてください。(図 2.55)



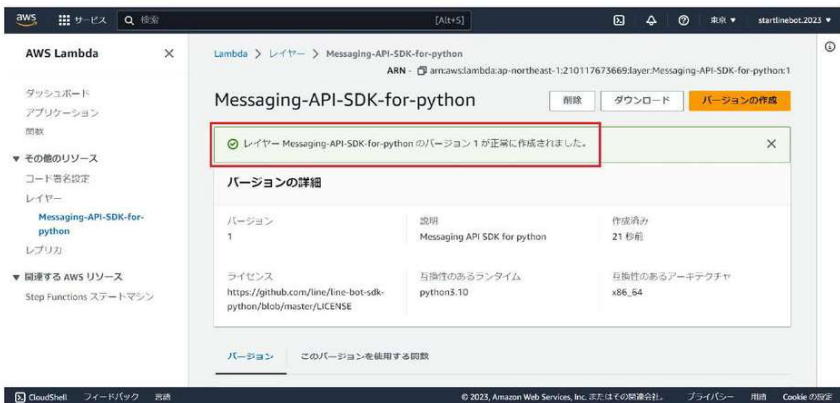
▲図 2.55 デスクトップにある python.zip を選択してアップロード

必要事項を入力したら「作成」をクリックします。(図 2.56)



▲図 2.56 必要事項を入力したら「作成」をクリックする

これで Messaging API SDK のレイヤーができました！（図 2.57）



▲図 2.57 レイヤーができた！

Lambda 関数を作成する

次は Lambda 関数を作成しますので、左メニューの「関数」から「関数の作成」を開いてください。（図 2.58）



▲図 2.58 「関数の作成」をクリックする

関数の作成に必要な情報は、次のように入力します。(表 2.2)

▼表 2.2 関数の作成

関数名	Bot-Server-on-Lambda
ランタイム	Python 3.10
アーキテクチャ	x86_64
その他の設定	すべてデフォルトのまま

必要事項を入力したら「関数の作成」をクリックします。(図 2.59)

関数の作成

AWS Serverless Application Repository applications have moved to [Create application](#).

☒ 一から作成
 シンプルな Hello World の例で開始します。

☐ 設計図の使用
 一般のユースケース用のサンプルコードと設定プリセットから Lambda アプリケーションを構築します。

☐ コンテナイメージ
 関数にデプロイするコンテナイメージを選択します。

基本的な情報

関数名
 関数の名称を入力します。
 Bot-Server-on-Lambda
文字、数字、ハイフン、アンダースコアのみ使用可能で、スペースは使用できません。

ランタイム
 関数の記述に使用する言語を選択します。コンソールコードエディタは Node.js、Python、および Ruby のみをサポートすることに注意してください。
 Python 3.10

アーキテクチャ
 関数コードに必要な命令セットアーキテクチャを選択します。
☒ x86_64
☐ arm64

アクセス権限
 デフォルトでは、Lambda は Amazon CloudWatch Logs にログをアップロードするアクセス許可を持つ実行ロールを生成します。このデフォルトのロールは、後で追加の権限を追加するとさらにカスタマイズできます。

▶ デフォルトの実行ロールの変更

▶ 詳細設定

キャンセル **関数の作成**

▲図 2.59 入力したら「関数の作成」をクリックする

これで Lambda 関数ができました！ 続いて、この Lambda 関数に、先に作っておいた Messaging API SDK のレイヤーを追加したいので、[Layers] をクリックします。(図 2.60)



▲図 2.60 Lambda 関数ができたら [Layers] をクリックする

[レイヤーの追加] をクリックします。(図 2.61)



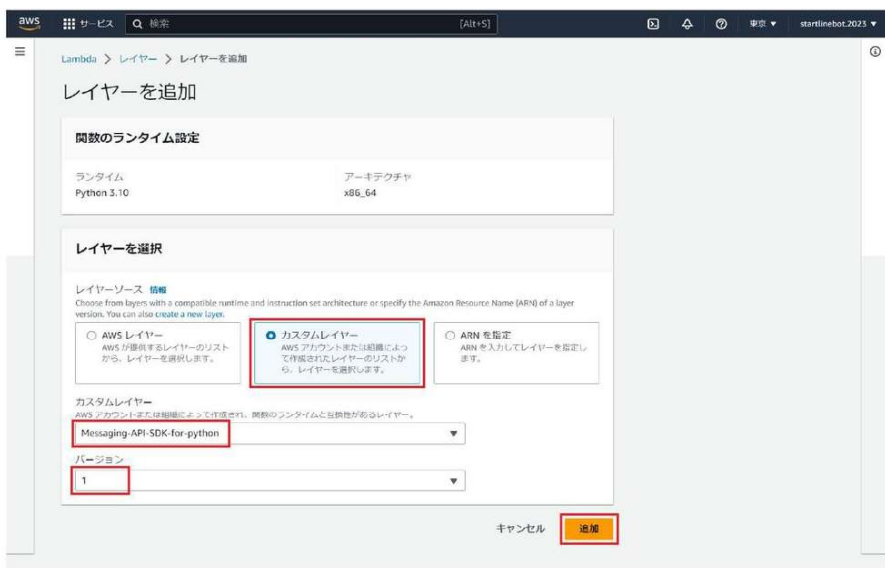
▲図 2.61 [レイヤーの追加] をクリックする

レイヤーは次のように選択します。(表 2.3)

▼表 2.3 レイヤーを選択

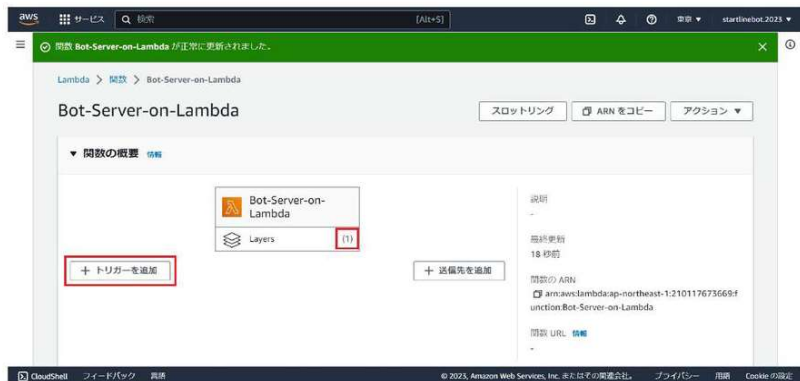
レイヤーソース	カスタムレイヤー
カスタムレイヤー	Messaging-API-SDK-for-python
バージョン	1

使用するレイヤーを選択したら「追加」をクリックします。(図 2.62)



▲図 2.62 「追加」をクリックする

[Layers] の後ろの数字が (0) から (1) になりました。これで Lambda 関数に Messaging API SDK のレイヤーが追加できました！ こうしてレイヤーを追加することで、Lambda 関数で Messaging API SDK が使えるようになります。(図 2.63)

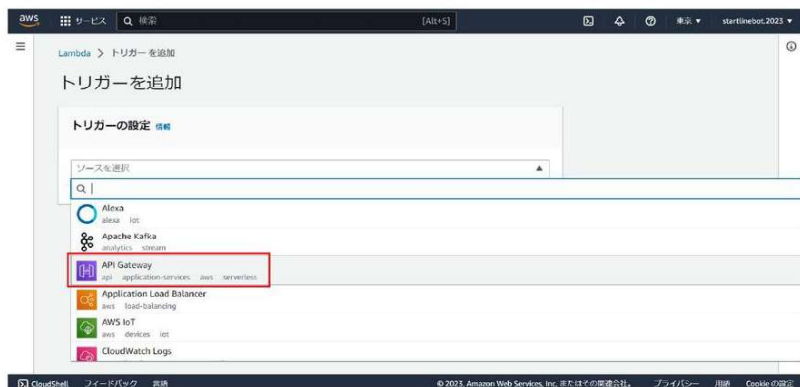


▲図 2.63 Messaging API SDK のレイヤーが追加できた！

次は API Gateway を作成しますので、[トリガーを追加] を開いてください。

API Gateway を作成する

[ソースを選択] で [API Gateway] を選択します。(図 2.64)



▲図 2.64 [API Gateway] を選択する

トリガーの設定は次のようにします。(表 2.4)

▼表 2.4 トリガーの設定

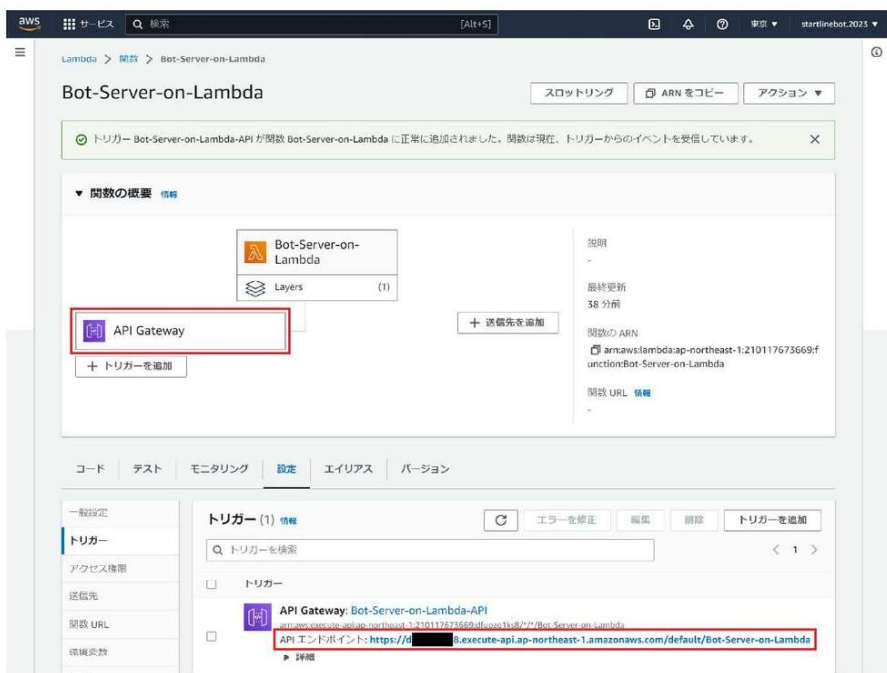
インテント	新規 API を作成
API タイプ	HTTP API
セキュリティ	開く
その他の設定	すべてデフォルトのまま

トリガーの設定を選択したら「追加」をクリックします。(図 2.65)



▲図 2.65 「追加」をクリックする

これで API Gateway が作成できました！（図 2.66）



▲図 2.66 API Gateway ができた！

「トリガー」の中にある「API エンドポイント」の URL をブラウザで開くと、AWS Lambda から「Hello from Lambda!」というレスポンスが返ってきます。（図 2.67）



▲図 2.67 「API エンドポイント」の URL を開くとレスポンスが返ってきた

この「API エンドポイント」の URL は、後で Webhook を受け止めるボットサーバーの「Webhook URL」として使用します。チャンネルアクセストークンやチャンネルシークレットと同様に、パソコンのメモ帳にしっかりメモしておいてください。

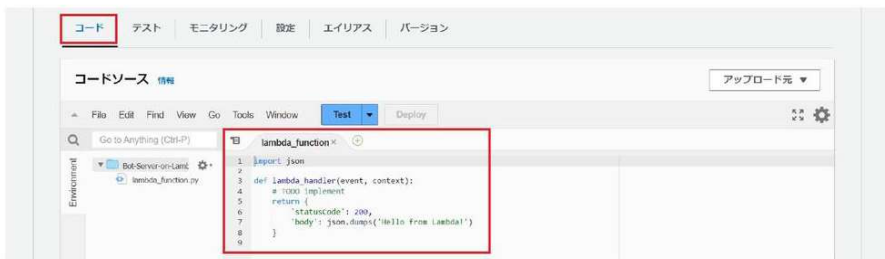
Lambda 関数で動かす python のコードを書く

ブラウザで「API エンドポイント」の URL を開いたとき、「Hello from Lambda!」というレスポンスが返ってきたのは、Lambda 関数の「コード」タブのコードソースに次のようなコードが書いてあったからです。（リスト 2.2）

▼リスト 2.2 Lambda 関数のデフォルトのコード

```
1: import json
2:
3: def lambda_handler(event, context):
4:     # TODO implement
5:     return {
6:         'statusCode': 200,
7:         'body': json.dumps('Hello from Lambda!')
8:     }
```

このデフォルトのコードは、リクエストが来たら、ステータスコード 200 と共に「Hello from Lambda!」というメッセージを含む JSON を返すようになっています。（図 2.68）



▲図 2.68 ステータスコード 200 と JSON を返すデフォルトのコード

このコードを、次のように Webhook を受け取ってオウム返しするコード^{*32}に書き直してみましょう。（リスト 2.3）

^{*32} このコードは GitHub にある本書のリポジトリで公開されています。PDF から直接コピー＆ペーストすると行番号などが混入してしまうため、ぜひこちらを使ってください。 <https://github.com/mochikoAsTech/startLINEBot/blob/master/articles/parrotbot.py>

▼リスト 2.3 Webhook でメッセージを受け取ってオウム返りするコード

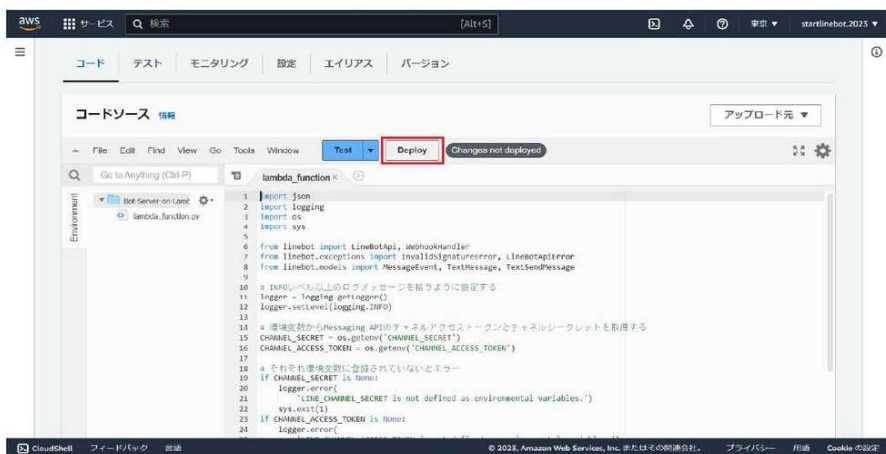
```
1: import json
2: import logging
3: import os
4: import sys
5:
6: from linebot import LineBotApi, WebhookHandler
7: from linebot.exceptions import InvalidSignatureError, LineBotApiError
8: from linebot.models import MessageEvent, TextMessage, TextSendMessage
9:
10: # INFOレベル以上のログメッセージを拾うように設定する
11: logger = logging.getLogger()
12: logger.setLevel(logging.INFO)
13:
14: # 環境変数からMessaging APIのチャネルアクセストークンとチャネルシークレットを取得する
15: CHANNEL_ACCESS_TOKEN = os.getenv('CHANNEL_ACCESS_TOKEN')
16: CHANNEL_SECRET = os.getenv('CHANNEL_SECRET')
17:
18: # それぞれ環境変数に登録されていないとエラー
19: if CHANNEL_ACCESS_TOKEN is None:
20:     logger.error(
21:         'LINE_CHANNEL_ACCESS_TOKEN is not defined as environmental variables.')
22:     sys.exit(1)
23: if CHANNEL_SECRET is None:
24:     logger.error(
25:         'LINE_CHANNEL_SECRET is not defined as environmental variables.')
26:     sys.exit(1)
27:
28: line_bot_api = LineBotApi(CHANNEL_ACCESS_TOKEN)
29: webhook_handler = WebhookHandler(CHANNEL_SECRET)
30:
31:
32: @webhook_handler.add(MessageEvent, message=TextMessage)
33: def handle_message(event):
34:
35:     # 応答トークンを使って回答を応答メッセージで送る
36:     line_bot_api.reply_message(
37:         event.reply_token, TextSendMessage(text=event.message.text))
38:
39:
40: def lambda_handler(event, context):
41:
42:     # リクエストヘッダーにx-line-signatureがあることを確認
43:     if 'x-line-signature' in event['headers']:
44:         signature = event['headers']['x-line-signature']
45:
46:         body = event['body']
47:         # 受け取ったWebhookのJSONを目視確認できるようにINFOでログに吐く
48:         logger.info(body)
49:
50:         try:
51:             webhook_handler.handle(body, signature)
52:         except InvalidSignatureError:
53:             # 署名を検証した結果、飛んできたのがLINEプラットフォームからのWebhookでなければ400を返す
```

```

54:         return {
55:             'statusCode': 400,
56:             'body': json.dumps('Only webhooks from the LINE Platform will
be accepted.')
57:         }
58:     except LineBotApiError as e:
59:         # 応答メッセージを送ろうとしたがLINEプラットフォームからエラーが返ってきたら
エラーを吐く
60:         logger.error('Got exception from LINE Messaging API: %s\n' % e.message)
61:         for m in e.error.details:
62:             logger.error(' %s: %s' % (m.property, m.message))
63:
64:     return {
65:         'statusCode': 200,
66:         'body': json.dumps('Hello from Lambda!')}
67: }

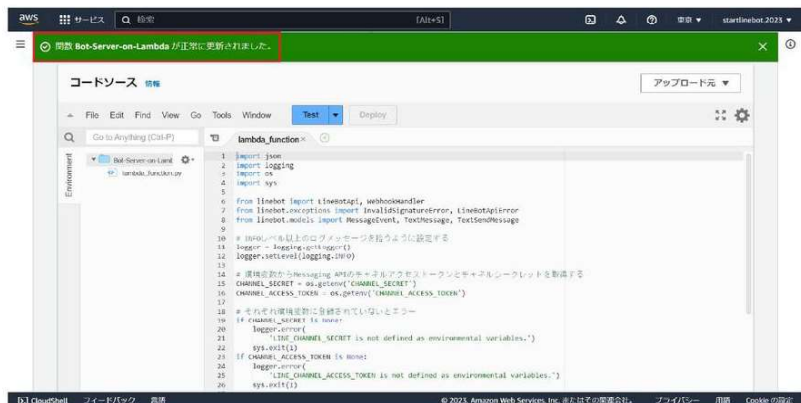
```

コードを直したら、[Deploy] を押してデプロイ（修正後のコードを反映）します。（図 2.69）



▲図 2.69 [Deploy] を押して修正後のコードを反映する

「関数 Bot-Server-on-Lambda が正常に更新されました。」と表示されたらデプロイ完了です。(図 2.70)



▲図 2.70 デプロイ完了

環境変数を設定する

いまデプロイしたコードを実際に動かすには、Messaging API のチャンネルアクセストークンとチャンネルシークレットが必要です。どちらもソースコードには直接書かず、環境変数として設定しておいて、コードからは `CHANNEL_ACCESS_TOKEN = os.getenv('CHANNEL_ACCESS_TOKEN')` というように環境変数を参照する形にしています。

Lambda 関数の「設定」タブから「環境変数」を開いて、「編集」をクリックします。(図 2.71)



▲図 2.71 「環境変数」の「編集」をクリックする

[環境変数の追加] を2回クリックして、[キー] と [値] を次のように設定します。チャネルアクセストークンとチャネルシークレットは、「2.4.1 LINE Developers コンソールでチャネルアクセストークンを発行する」でコピーしてメモ帳に保存してあるはずです。(表 2.5)

▼表 2.5 環境変数の編集

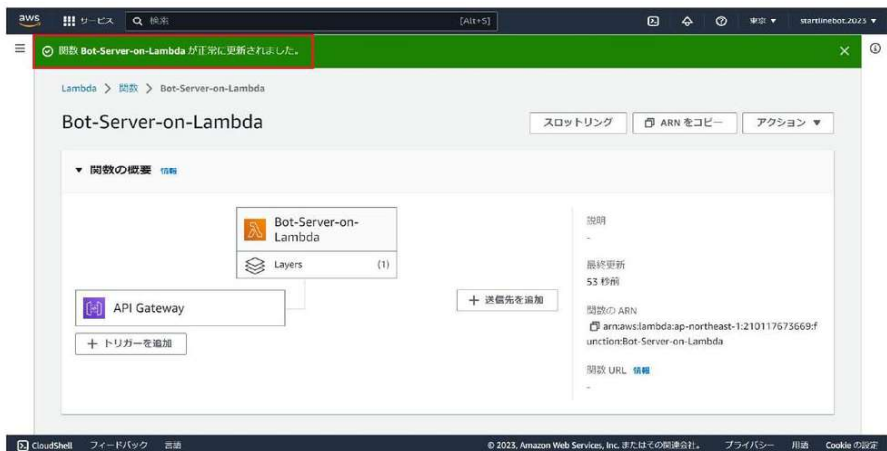
キー	値
CHANNEL_ACCESS_TOKEN	チャネルアクセストークン
CHANNEL_SECRET	チャネルシークレット

チャネルアクセストークンとチャネルシークレットを環境変数に追加したら [保存] をクリックします。(図 2.72)



▲図 2.72 環境変数を追加したら [保存] をクリックする

「関数 Bot-Server-on-Lambda が正常に更新されました。」と表示されたら環境変数の設定完了です。(図 2.73)



▲ 図 2.73 環境変数の設定完了

これでボットサーバーの準備は万端です。LINE Developers コンソールに戻って、ボットサーバー（「API エンドポイント」の URL）を「Webhook URL」に設定しましょう。

2.6.3 Webhook URL を設定する

LINE Developers コンソール^{*33}を開いて、Messaging API チャネルの [Messaging API 設定] タブにある [Webhook 設定] の [Webhook URL] を登録します。[編集] をクリックしてください。(図 2.74)



▲図 2.74 [Webhook URL] の [編集] をクリックする

「API Gateway を作成する」でメモしておいた API エンドポイントの URL を貼り付けて、[更新] をクリックしてください。(図 2.75)



▲図 2.75 API エンドポイントの URL を貼り付けて [更新] をクリックする

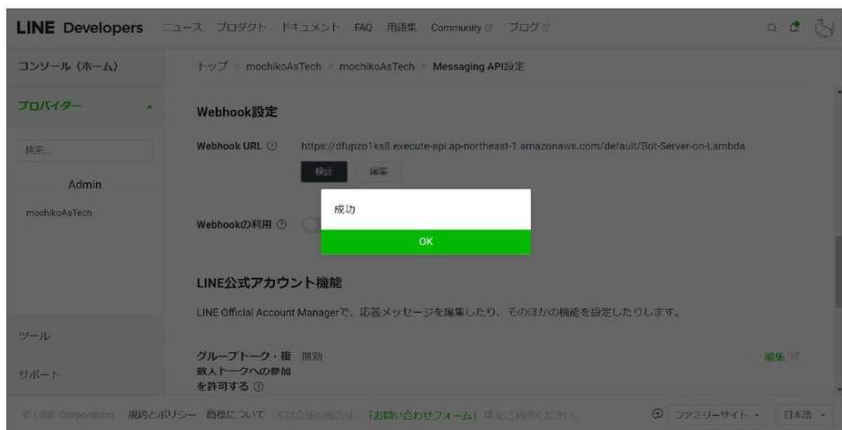
^{*33} <https://developers.line.biz/console/>

Webhook URL を設定したら、[検証] を押してボットサーバーの動作検証をしてみましょう。(図 2.76)



▲図 2.76 [検証] を押す

[成功] と返ってきたら、LINE プラットフォームからの Webhook をボットサーバーが受け取って、ちゃんとステータスコード 200 を返してきています。[OK] を押してポップアップを閉じておきましょう。やりましたね！ おめでとうございます。(図 2.77)



▲図 2.77 [成功] と返ってきたら [OK] をクリックする

検証ではうまくいきましたが、今後ボットサーバーが Webhook をきちんと受け取れなかったときに備えて、[Webhook URL] の少し下にある [エラーの統計情報]^{*34}をオンにしておきましょう。(図 2.78)



▲図 2.78 [エラーの統計情報] をオンにしておく

この [エラーの統計情報] をオンにしておくと、もしボットサーバーが Webhook の受け取りに失敗した場合に、LINE Developers コンソール上でそのログが確認できます。Webhook URL の [検証] を押してエラーが返ってきたときや、友だちがメッセージを送ってきたのに LINE 公式アカウントからきちんと返信が送れていない場合は、ボットサーバーのログ^{*35}と共に、[統計情報] タブでエラーの統計情報も確認しましょう。

^{*34} Webhook の送信におけるエラーの統計情報を確認する | LINE Developers <https://developers.line.biz/ja/docs/messaging-api/receiving-messages/#error-statistics-aggregation>

^{*35} ボットサーバーのログについては、「2.6.5 LINE プラットフォームから飛んできた Webhook を目視確認する」で後述します。

[検証] を押して [成功] と返ってきたことから、ボットサーバーが問題なく動いていると分かったので、同じ [Messaging API 設定] タブにある [応答メッセージ] の [編集] から LINE Official Account Manager を開きます。(図 2.79)



▲図 2.79 [応答メッセージ] の [編集] から LINE Official Account Manager を開く

ユーザーがメッセージを送ってきたら、今後は Webhook を受け取ったボットサーバーから応答したいので、[応答設定] の [Webhook] をオンにして、代わりに [応答メッセージ] をオフにしてください。(図 2.80)



▲図 2.80 [Webhook] をオンにして [応答メッセージ] をオフにする

これで Webhook の設定は完了です。LINE プラットフォームからの Webhook が、AWS Lambda で用意したボットサーバーに向かって飛んでくるようになりました。

2.6.4 友だち追加されたときのあいさつメッセージを併用する

友だち追加されたときに、自動で任意のメッセージを送れる「あいさつメッセージ」も、LINE Official Account Manager の「応答設定」でオン、オフの設定ができます。このあいさつメッセージは、Webhook と併用することが可能です。本書では、友だち追加されたときの応答は「あいさつメッセージ」で行い、メッセージが届いたときの応答はボットサーバーから行いたいので、あいさつメッセージはオンのままで構いません。

なお友だち追加されたときに Webhook で飛んでくるフォローイベント^{*36}を用いると、「あいさつメッセージ」と同等の処理をボットサーバーでも実現できます。

2.6.5 LINE プラットフォームから飛んできた Webhook を目視確認する

先ほど、LINE Developers コンソールで Webhook URL の「検証」を押したときに、LINE プラットフォームからボットサーバーに飛んできた Webhook を目視確認してみましょう。ボットサーバーのログは、AWS の CloudWatch で確認できます。AWS のマネジメントコンソールを開いて、上部の検索窓で **cloudwatch** と検索し、CloudWatch を開きます。(図 2.81)



▲図 2.81 検索窓から CloudWatch を開く

CloudWatch を開いたら、左メニューの「ロググループ」から `[/aws/lambda/Bot-`

^{*36} フォローイベント | LINE Developers

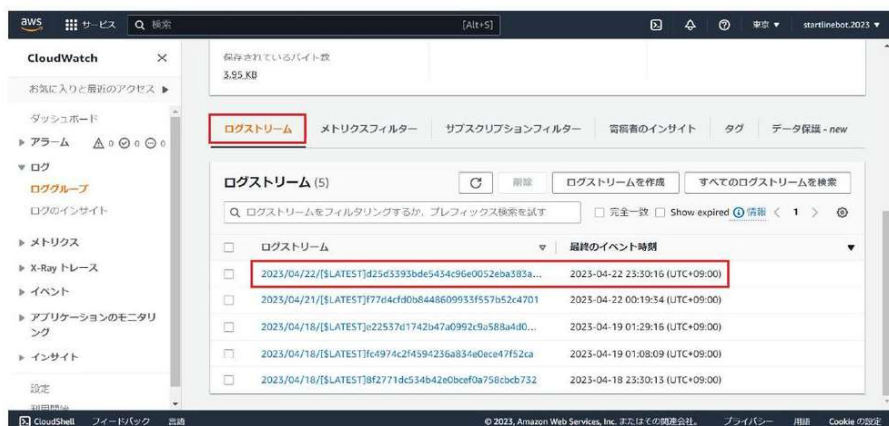
<https://developers.line.biz/ja/reference/messaging-api/#follow-event>

Server-on-Lambda] を開きます。(図 2.82)



▲図 2.82 [/aws/lambda/Bot-Server-on-Lambda] を開く

いちばん上にある最新のリクエストのログストリームを開きます。(図 2.83)



▲図 2.83 いちばん上にあるログストリームを開く

[INFO] からはじまる行を開いて確認します。すると、Webhook を受け取ってオウム返しのコード (リスト 2.3) の 48 行目で出力しておいたログが確認できます。(図 2.84)



▲図 2.84 LINE プラットフォームから届いた Webhook の JSON が確認できる

これが LINE プラットフォームから届いた Webhook の JSON です。(リスト 2.4)

▼リスト 2.4 [検証] を押したときに飛んできた Webhook の JSON

```
1: {
2:   "destination": "U61 (中略) a19",
3:   "events": []
4: }
```

destination には、その「destination (お届け先)」という名前のとおり、LINE プラットフォームが Webhook を渡す相手、つまり LINE 公式アカウント自身のユーザー ID が入っています。

今回は動作検証だけだったので **events** の中身は空配列でしたが、友だち追加されたときや、友だちからメッセージが届いたときは、ここにメッセージイベントやフォローイベントなどが入ってきます。**events** の中に入ってくる Webhook イベントオブジェクトについては、公式ドキュメントの API リファレンス^{*37}を参照してください。

^{*37} Webhook イベントオブジェクト | Messaging API リファレンス | LINE Developers <https://developers.line.biz/ja/reference/messaging-api/#webhook-event-objects>

2.6.6 LINE 公式アカウントに話しかけてオウム返しを確認しよう

それでは LINE で LINE 公式アカウントに話しかけて、まったく同じメッセージがオウム返しされるか確認してみましょう。(図 2.85)



▲図 2.85 メッセージを送ったら LINE 公式アカウントがオウム返ししてくれた

おめでとうございます！ オウム返ししてくれる LINE Bot の完成です。「LINE Bot をつくる」というアチーブメントを達成です！

【コラム】これは本当に LINE プラットフォームから来た Webhook ？

ボットサーバーに LINE プラットフォームから Webhook が届いたら、その内容に応じて返信を送ったり、何か処理をしたりします。

このとき、届いたリクエストが本当に LINE プラットフォームから届いた Webhook なのか、それとも LINE プラットフォームを装った第三者からの攻撃なのか、どうやって判別すればよいのでしょうか？

アクセス元の IP アドレスが分かれば、IP アドレスで制限をかけることで、LINE プラットフォーム以外からのアクセスを遮断できますが、残念ながら LINE プラットフォームは IP アドレスのレンジを開示していません。^{*38}代わりに推奨されているのが「署名の検証」という方法です。

LINE プラットフォームから届く Webhook には、そのリクエストヘッダーに必ず **x-line-signature** という署名が含まれています。Messaging API チャネルのチャンネルシークレットを秘密鍵として扱い、届いた Webhook のリクエスト

ボディのダイジェスト値を取得し、さらにそのダイジェスト値をチャンネルシークレットを用いて Base64 エンコードした値と、リクエストヘッダーの **x-line-signature** の署名が一致することを確認できれば、これが本当に LINE プラットフォームから届いた Webhook である、というセキュリティの担保ができます。

この検証をしないで、無条件に「テキストメッセージの Webhook イベントが届いたらユーザーに返信する」のような処理をしていると、悪意のある第三者がボットサーバーに偽の Webhook を投げ続けることで、LINE Bot を介して特定のユーザーに大量のメッセージを送りつける攻撃が可能になってしまいます。

署名検証の各言語ごとのコードサンプルは、公式ドキュメントの「署名を検証する^{*39}」にありますし、SDK を用いることで簡単に検証できます。

Webhook を受け取ってオウム返しするコード（リスト 2.3）では、リクエストヘッダーに含まれていた **x-line-signature** を 44 行目で **signature** に詰めて、51 行目で署名検証しています。署名検証した結果、もし LINE プラットフォームを装った第三者からのリクエストだったら、52 行目からのエラー処理でステータスコード 400 を返して、返信の処理は行わないようになっています。

^{*38} Webhook | LINE Developers

<https://developers.line.biz/ja/reference/messaging-api/#webhooks>

^{*39} 署名を検証する | LINE Developers

<https://developers.line.biz/ja/reference/messaging-api/#signature-validation>

2.7 OpenAI の API を使った AI チャットボットを作ってみよう

LINE 公式アカウントにメッセージを送って、無事にオウム返し of チャットボットが動いてくれました。うれしいね！*40 うれしいものの、「送ったメッセージをオウム返しされた…けど…だからなに？」という気持ちにもなるので、今度は OpenAI の API を使って、LINE 公式アカウントの中身をちゃんと役に立つ AI チャットボットに作り変えてみましょう。

2.7.1 ChatGPT と GPT-3.5 とは

ChatGPT は OpenAI が提供している対話型のウェブサービスです。OpenAI のアカウントを作れば、誰でも利用できます。(図 2.86)

- ChatGPT - OpenAI
– <https://chat.openai.com/>



▲図 2.86 ChatGPT (GPT-3.5) に ChatGPT のことを質問している様子

*40 Oh!バモさん、オウム返し watch…ウレシイウレシイね。セイジ先生の授業はとてもよいので一通り受けることをお勧めする。

この ChatGPT のベースとして応答を生成している言語モデルが GPT-3.5、およびその後継である GPT-4^{*41}です。(図 2.87)



▲図 2.87 ChatGPT (GPT-4) に ChatGPT のことを質問している様子

それでは先ほどのオウム返しボットを少し作り変えて、ユーザーが LINE で質問を送ると GPT-3.5 がその文脈に基づいて回答を生成し、その回答が LINE 公式アカウントからのメッセージとして返ってくる、という AI チャットボットにしていきたいと思います。^{*42}

2.7.2 OpenAI に登録してシークレットキーを取得する

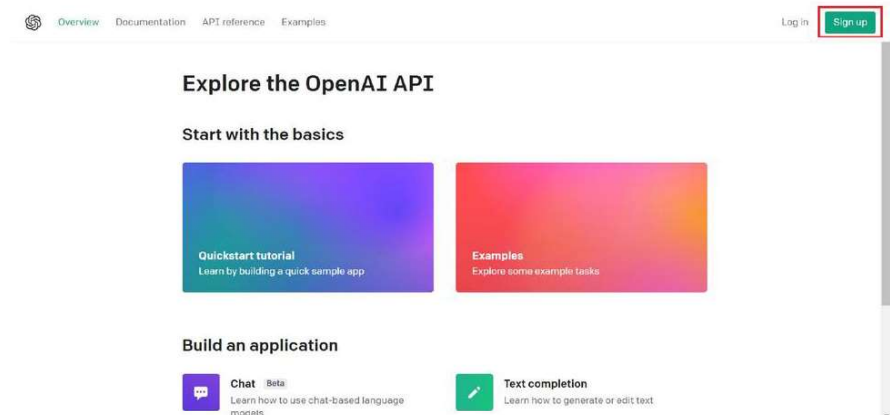
Messaging API を使うにはチャンネルアクセストークンが必要だったように、OpenAI API を使うには、OpenAI API のサイトでアカウントを作ってシークレットキーを取得する必要があります。

それでは OpenAI API のサイトにアクセスして、右上の [Sign up] を開いてください。(図 2.88)

- OpenAI API
 - <https://platform.openai.com/overview>

^{*41} 2023 年 5 月時点では、ChatGPT Plus という月額 20 ドルのサブスクリプションプランを契約すると言語モデルとして GPT-4 を選択できます。契約せずに無料で使う場合は、デフォルトの GPT-3.5 しか選べません。

^{*42} 本書を書いている最中に「ChatGPT だー!」「GPT-3.5 だー!」「いや GPT-4 だー!」とインターネットがお祭り騒ぎになったので、急いで動作検証をして、この「2.7 OpenAI の API を使った AI チャットボットを作ってみよう」という節を書き足しました。なんとか間に合ってよかった…!



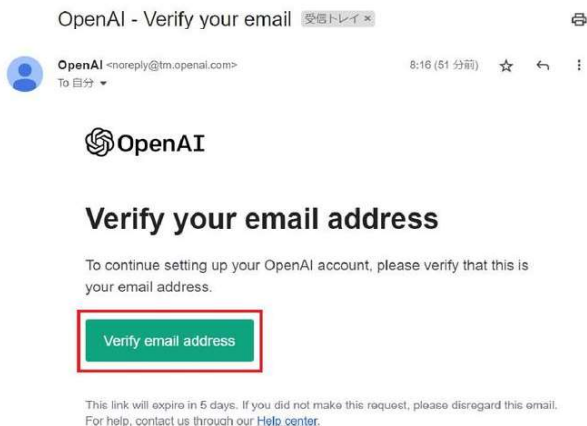
▲図 2.88 右上の [Sign up] を開く

[Create your account] と表示されたら、メールアドレスを入力して [Continue] をクリックします。(図 2.89)

続いてパスワードを入力して [Continue] をクリックすると、[Verify your email] と表示されます。

▲図 2.89 メールアドレスとパスワードを入力して [Continue] をクリックする

すると、入力したメールアドレス宛てに [OpenAI - Verify your email] という件名のメールが届きます。[Verify email address] をクリックしてください。(図 2.90)



▲図 2.90 届いたメールの [Verify email address] をクリックする

メールアドレスの確認ができれば、[Tell us about you] と表示されます。苗字と名前、そして生年月日を入力して、[Continue] をクリックします。(図 2.91)

[Verify your phone number] と表示されたら、携帯電話の電話番号を入力します。プラス記号と国番号の 81 から始まる国際的な電話番号の形式なので、あなたの電話番号が「080-0123-4567」なら「+81」の後ろに「8001234567」と入力してください。入力したら [Send code] をクリックします。

Tell us about you

Organization name (optional)

>

Verify your phone number

● ▼

+81 80 [REDACTED]

By clicking "Continue", you agree to our [Terms and acknowledge our Privacy policy](#)

▲図 2.91 苗字と名前と生年月日、電話番号を入力する

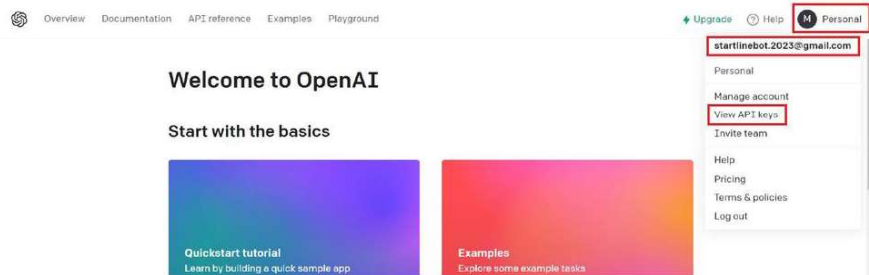
すると入力した電話番号のスマートフォン宛てに、[あなたの OpenAI API 認証コード] という SMS が届きます。(図 2.92)

SMS に書いてある 6 桁の認証コードを、[Enter code] と表示された画面で入力してください。



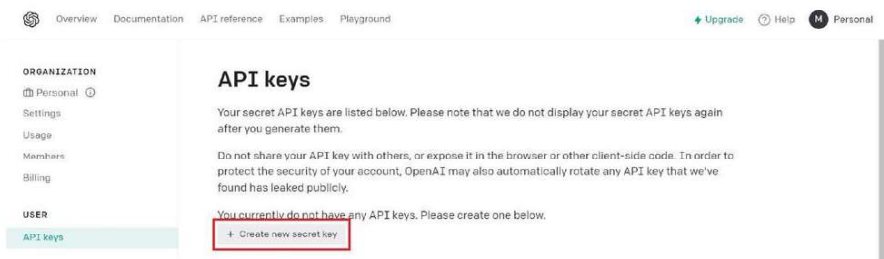
▲図 2.92 SMS で届いた 6 桁の認証コードを [Enter code] の画面で入力する

右上のアイコンをクリックして自分のメールアドレスが表示されていたら、OpenAI API のアカウント登録は完了です！ おめでとうございます！ それでは [View API keys] を開いてください。(図 2.93)



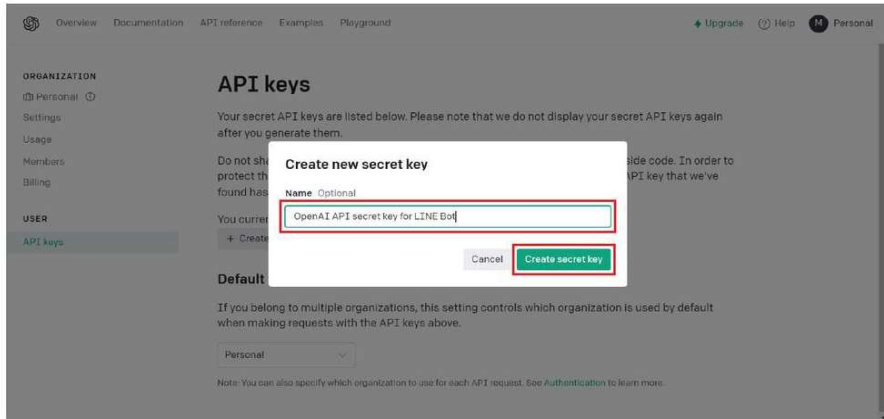
▲図 2.93 OpenAI API のアカウント登録が完了したら [View API keys] を開く

OpenAI API をたたくためのシークレットキーというものが欲しいのですが、まだ存在していないので [Create new secret key] をクリックして作成しましょう。(図 2.94)



▲図 2.94 [Create new secret key] をクリックする

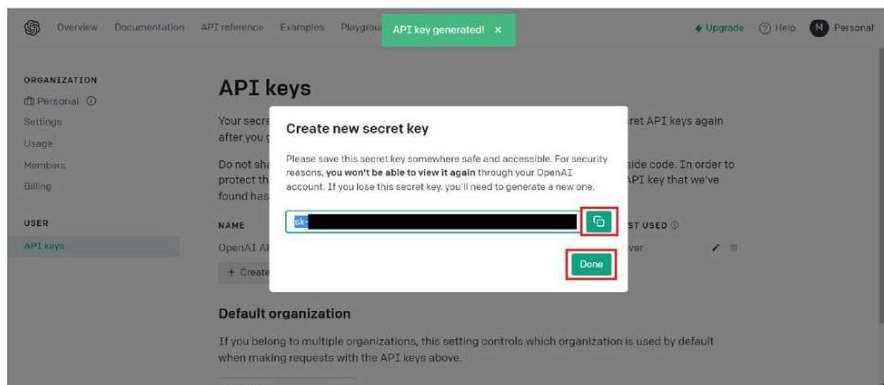
シークレットキーの [Name] は任意入力ですが、何の用途で作った鍵なのか後で分からなくなってしまうように、**OpenAI API secret key for LINE Bot** と書いておきましょう。[Create secret key] をクリックします。(図 2.95)



▲図 2.95 [Create secret key] をクリックする

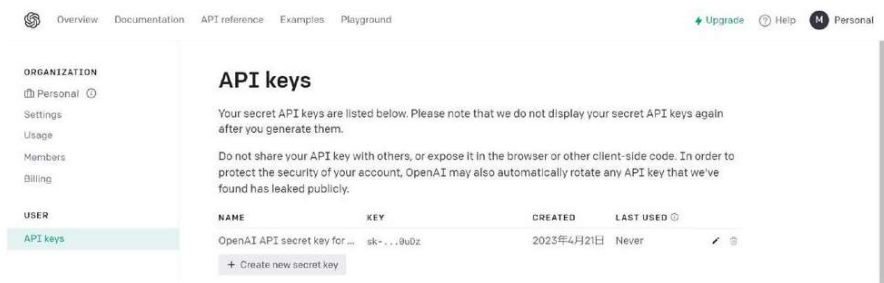
シークレットキーが作成されました。このシークレットキーはこの画面でコピーしておかないと、以降二度と表示されません。もし無くしたらシークレットキーを再び作成することになります。コピーボタンを押して、チャンネルアクセストークンやチャンネルシーク

レットと同様に、パソコンのメモ帳にしっかりメモしておいてください。メモできたら [Done] をクリックして、ポップアップを閉じます。(図 2.96)



▲図 2.96 シークレットキーは二度と表示されないのでしっかりメモしておこう

これで OpenAI API をたたくためのシークレットキーが手に入りました。(図 2.97)



▲図 2.97 シークレットキーが手に入った！

LINE Developers コンソールで取得したチャネルアクセストークンと同様に、このシークレットキーは OpenAI API をたたくときに身分証のような役割を果たします。うっかりシークレットキーが載った画面をブログで公開したり、ソースコードに直接書いて GitHub に Push したりしないように注意してください。

2.7.3 OpenAI API に質問を投げて回答を取得する

「2.4.2 Messaging API でブロードキャストメッセージを送信する」で curl コマンドを使って Messaging API をたたき、メッセージを送信したように、今度は curl コマンドで OpenAI API に質問を投げて回答を取得してみましょう。

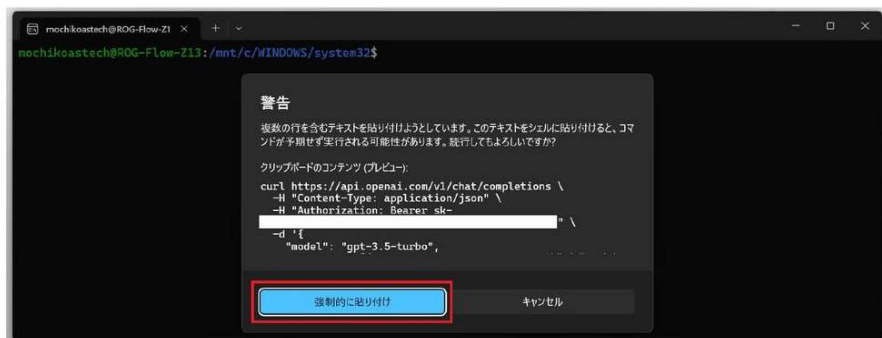
再び WSL またはターミナルを起動します。次の curl コマンド^{*43}（リスト 2.5）の 3 行目にある{**シークレットキー**}の部分、OpenAI API のシークレットキーに置き換えてください。シークレットキーは、ついさっき「2.7.2 OpenAI に登録してシークレットキーを取得する」でコピーしてメモ帳に保存しましたね。

▼リスト 2.5 curl コマンドで質問の回答を取得する

```
1: curl https://api.openai.com/v1/chat/completions \
2:   -H "Content-Type: application/json" \
3:   -H "Authorization: Bearer {シークレットキー}" \
4:   -d '{
5:     "model": "gpt-3.5-turbo",
6:     "messages": [{"role": "user", "content": "技術書典ってなんですか？"}]
7:   }'
```

シークレットキーを置き換えたら、curl コマンドをまるごとコピーして WSL もしくはターミナルに貼り付けます。WSL の場合は、複数行をまとめて貼り付けると警告が出ますが、[強制的に貼り付け] を押します。（図 2.98）

^{*43} このコードは GitHub にある本書のリポジトリで公開されています。PDF から直接コピー＆ペーストすると行番号などが混入してしまうため、ぜひこちらを使ってください。 <https://github.com/mochikoAsTech/startLINEBot/blob/master/articles/ask-openai-question.sh>



▲図 2.98 複数行の貼り付けに対する警告が出たら「強制的に貼り付け」を押す

貼り付けたら Enter を押して curl コマンドを実行します。(図 2.99)



▲図 2.99 貼り付けたら Enter を押して実行する

回答の生成には時間がかかりますが、少し経つとレスポンスが画面に出力されます。curl コマンドを使って OpenAI API に質問を投げた結果、レスポンスとして回答を含む JSON が返ってきました！(図 2.100)

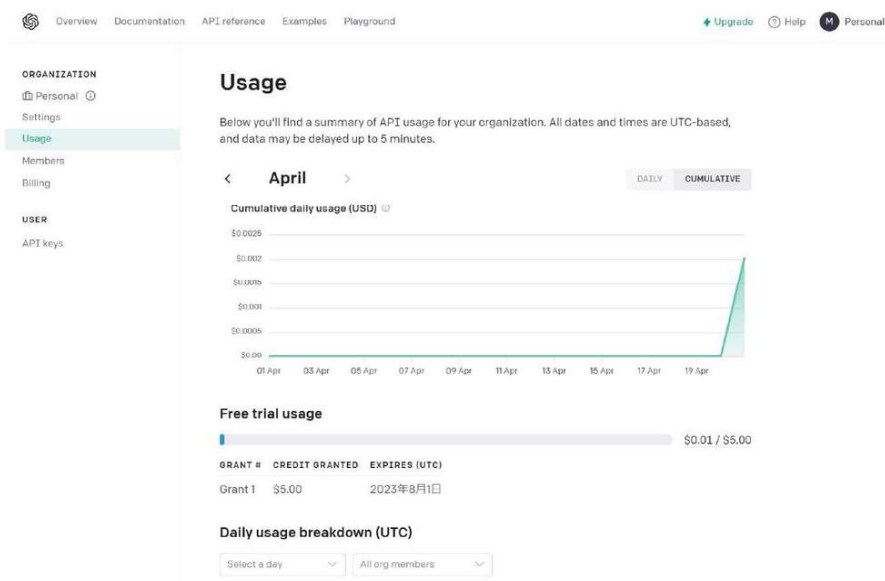


▲図 2.100 回答を含む JSON が返ってきた

リクエストで指定した **role** や **content**、返ってきたレスポンスの値がそれぞれ何を表しているのかは、OpenAI API の API リファレンス^{*44}を参照してください。

curl コマンドで OpenAI API に質問を投げて、回答を取得できましたね。「API をたたく」ことに少し慣れてきましたか？

ちなみに 2023 年 5 月現在、OpenAI API はアカウントを作ってから最初の 3 ヶ月は 5 ドル分まで無料^{*45}で使えるようです。自分がすでに無料枠をどれくらい使ったのかは、OpenAI API の Usage^{*46}で確認できます。(図 2.101)



▲図 2.101 無料枠をどれくらい使ったのかは Usage で確認しよう

では続いて、OpenAI API の SDK の準備をしていきましょう。

^{*44} Create chat completion | API Reference - OpenAI API <https://platform.openai.com/docs/api-reference/chat/create>

^{*45} Pricing <https://openai.com/pricing>

^{*46} Usage - OpenAI API <https://platform.openai.com/account/usage>

2.7.4 OpenAI API の SDK を準備する

先ほどの Messaging API の SDK と同じように、OpenAI API も公式が Python と Node.js で SDK を用意してくれています。AWS Lambda で使うために、OpenAI API の python.zip を作成しましょう。

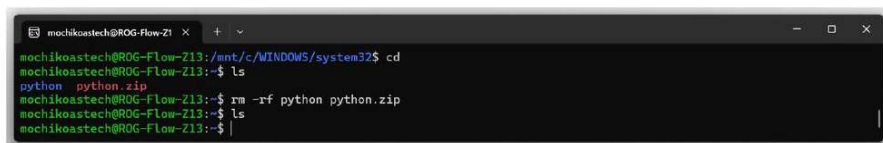
- Python library - OpenAI API
 – <https://platform.openai.com/docs/libraries/python-library>

Windows の場合

「2.4.2 Messaging API でブロードキャストメッセージを送信する」で使用した WSL を再び起動して、次のコマンドを順番にたたいていきます。\$ はプロンプトを表していますので入力しないでください。

まずは cd コマンドで Messaging API の SDK を準備したときと同じホームディレクトリに移動して、ls コマンドで python ディレクトリと python.zip が残っていることを確認します。rm コマンドで python ディレクトリと python.zip を削除したら、再び ls コマンドをたたいてもう無いことを確認します。(図 2.102)

```
$ cd
$ ls
$ rm -rf python python.zip
$ ls
```



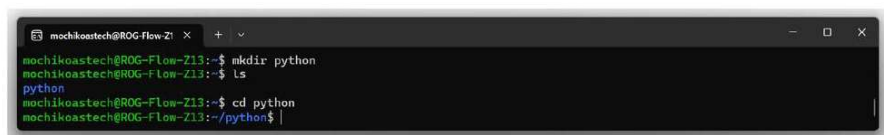
▲図 2.102 残っていた python フォルダと python.zip を削除する

そして再び mkdir コマンドで python^{*47} というディレクトリを作ります。ls コマンドで確認して「python」と表示されたら、問題なく python ディレクトリが作成できてい

^{*47} この作成するディレクトリの名前は必ず python にしてください。ディレクトリ名を python 以外にするとこの後の手順で正常に動きません。

ますので、作成した python ディレクトリの中に cd コマンドで移動してください。(図 2.103)

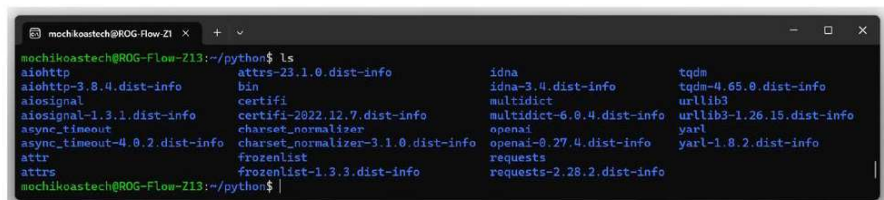
```
$ mkdir python
$ ls
$ cd python
```



▲図 2.103 python ディレクトリを作ってその中に移動する

いよいよ pip コマンドで SDK をパソコンの中を取得してきます。SDK を取ってきたら、ls コマンドで python ディレクトリの中身を確認してみましょう。中身がなんだかたくさん入っていれば成功です。(図 2.104)

```
$ pip install openai -t . --no-user
$ ls
```



▲図 2.104 取ってきた SDK が python ディレクトリの中にみっちり入っている

それでは cd コマンドで 1 つの上のディレクトリに移動して、zip コマンドで python ディレクトリをぎゅっと圧縮して ZIP に固めましょう。ls コマンドで python.zip と python ディレクトリが確認できれば OK です。(図 2.38)

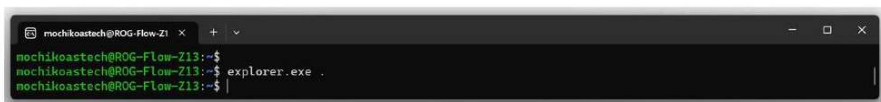
```
$ cd ..  
$ zip -r python.zip python  
$ ls
```



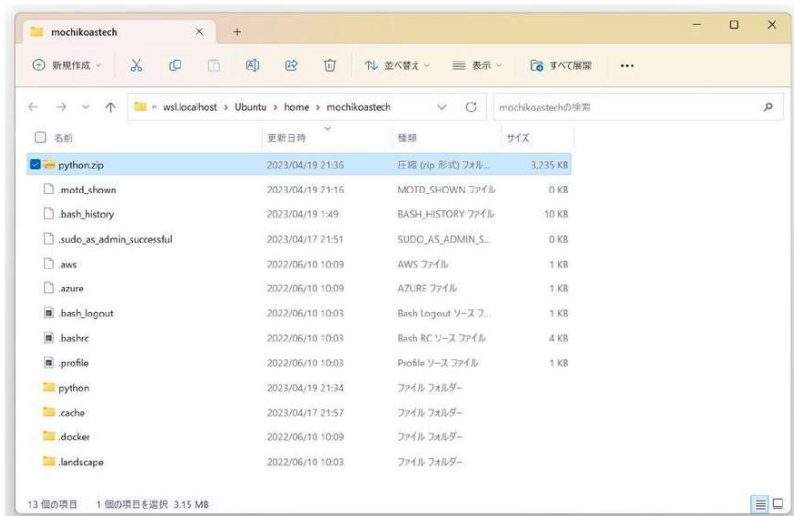
▲図 2.105 zip コマンドで python ディレクトリを ZIP に固めて ls コマンドで確認

最後に `explorer.exe .` をたたくと、WSL で見ていたディレクトリがエクスプローラで表示されます。(図 2.39、図 2.40)

```
$ explorer.exe .
```

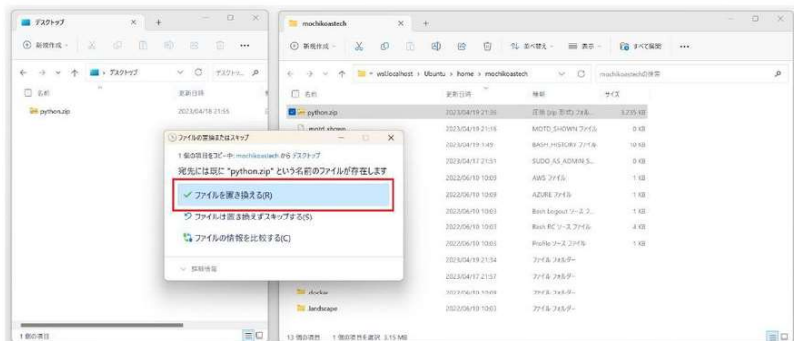


▲図 2.106 「explorer.exe .」をたたく



▲図 2.107 するとエクスプローラで python.zip のあるフォルダが表示される

作成した python.zip はこの後すぐに使うので、デスクトップに「ファイルを置き換える」でコピーしておきましょう。(図 2.108)



▲図 2.108 python.zip はデスクトップに「ファイルを置き換える」でコピーしておく

これで OpenAI API の SDK が準備できました。「2.7.5 OpenAI API SDK のレイヤーを作成する」に進んでください。

Mac の場合

「2.4.2 Messaging API でブロードキャストメッセージを送信する」で使用したターミナルを再び起動して、次のコマンドを順番にたたいていきます。`$`はプロンプトを表していますので入力しないでください。

まずは `ls` コマンドで、さきほど Messaging API の SDK を準備した `python` ディレクトリと `python.zip` が残っていることを確認します。`rm` コマンドで `python` ディレクトリと `python.zip` を削除したら、再び `ls` コマンドをたたいてもう無いことを確認します。(図 2.109)

```
$ ls -d python python.zip
$ rm -rf python python.zip
$ ls
```

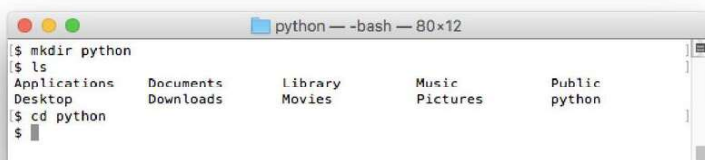


▲図 2.109 残っていた `python` ディレクトリと `python.zip` を削除する

そして再び `mkdir` コマンドで `python`^{*48}というディレクトリを作ります。`ls` コマンドで確認して「`python`」と表示されたら、問題なく `python` ディレクトリが作成できていますので、作成した `python` ディレクトリの中に `cd` コマンドで移動してください。(図 2.110)

^{*48} この作成するディレクトリの名前は必ず `python` にしてください。ディレクトリ名を `python` 以外にするとこの後の手順で正常に動きません。

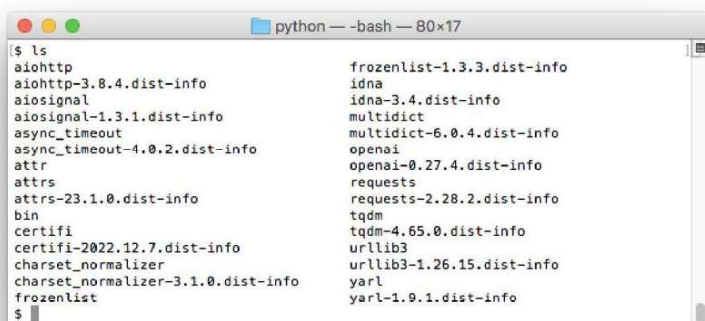

```
$ mkdir python
$ ls
$ cd python
```



▲図 2.110 python ディレクトリを作ってその中に移動する

pip3 コマンドで SDK をパソコンの中を取得してきます。SDK を取ってきたら、ls コマンドで python ディレクトリの中身を確認してみましょう。中身がなんだかたくさん入ってれば成功です。(図 2.111)

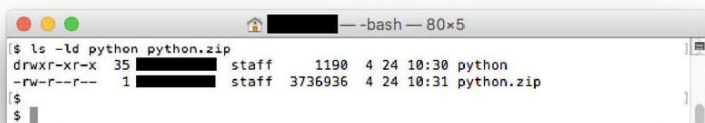
```
$ pip3 install openai -t . --no-user
$ ls
```



▲図 2.111 取ってきた SDK が python ディレクトリの中にみっちり入っている

取ってきた SDK をぎゅっと圧縮して ZIP に固めたいので、cd コマンドで1つの上のディレクトリに移動しましょう。zip コマンドで python ディレクトリをぎゅっと ZIP に固めます。ls コマンドで python.zip と python ディレクトリが確認できたら、これで SDK は準備完了です。(図 2.112)

```
$ cd ..  
$ zip -r python.zip python  
$ ls -ld python python.zip
```



```
python@python ~ % ls -ld python python.zip  
drwxr-xr-x  35 python  staff   1190  4 24 10:30 python  
-rw-r--r--   1 python  staff 3736936  4 24 10:31 python.zip  
python@python ~ %
```

▲図 2.112 zip コマンドで python ディレクトリを ZIP に固めて ls コマンドで確認

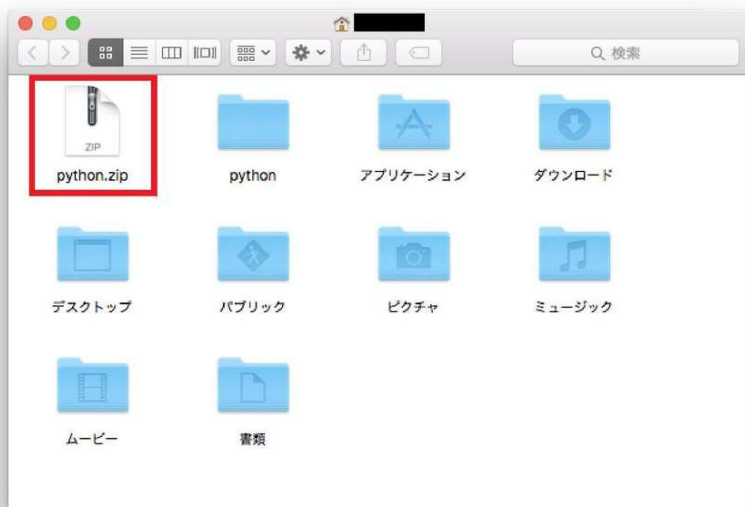
最後に open . をたたくと、ターミナルで見ていたディレクトリが Finder で表示されます。

```
open .
```



```
python@python ~ % open .
```

▲図 2.113 「open .」をたたく



▲図 2.114 すると Finder で python.zip のあるディレクトリが表示される

作成した python.zip はこの後すぐに使うので、デスクトップにコピーしておきましょう。これで OpenAI API の SDK が準備できました。

2.7.5 OpenAI API SDK のレイヤーを作成する

先ほどの Messaging API SDK と同様に、OpenAI API SDK のレイヤーを AWS Lambda に追加します。AWS のマネジメントコンソールから AWS Lambda を開いたら、左メニューの「レイヤー」から、「レイヤーの作成」をクリックします。(図 2.115)



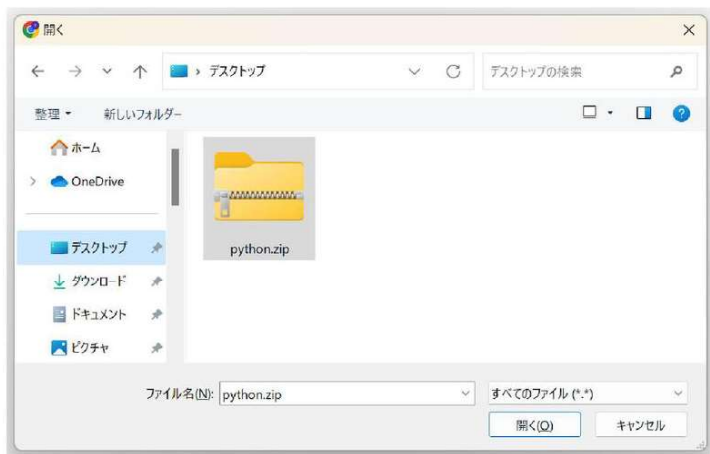
▲図 2.115 「レイヤーの作成」をクリック

「名前」と「説明」は次のように入力します。(表 2.6)

▼表 2.6 レイヤー設定

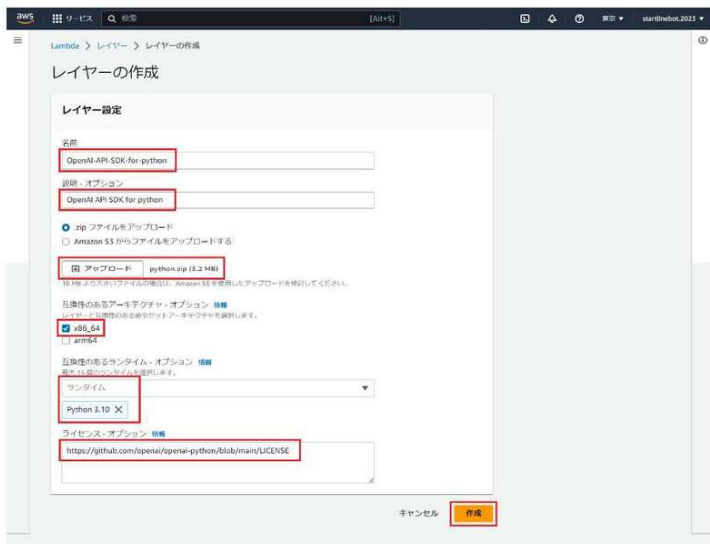
名前	OpenAI-API-SDK-for-python
説明	OpenAI API SDK for python
アップロード方法	zip ファイルをアップロード
アップロードするファイル	「2.7.4 OpenAI API の SDK を準備する」で用意した python.zip
互換性のあるアーキテクチャ	x86_64 にチェックを入れる
互換性のあるランタイム	Python 3.10 を選択
ライセンス	https://github.com/openai/openai-python/blob/main/LICENSE

「アップロード」をクリックしたら、「2.7.4 OpenAI API の SDK を準備する」で準備しておいた、デスクトップの python.zip を選択し、アップロードしてください。(図 2.116)



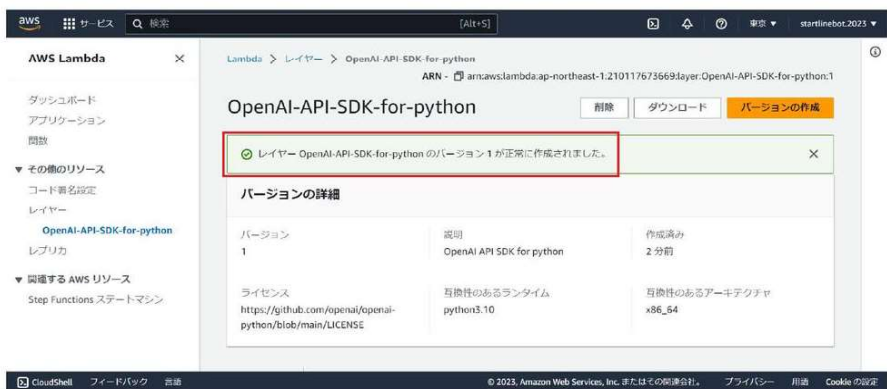
▲図 2.116 デスクトップにある python.zip を選択してアップロード

必要事項を入力したら「作成」をクリックします。(図 2.117)



▲図 2.117 必要事項を入力したら「作成」をクリックする

これで OpenAI API SDK のレイヤーができました！（図 2.118）

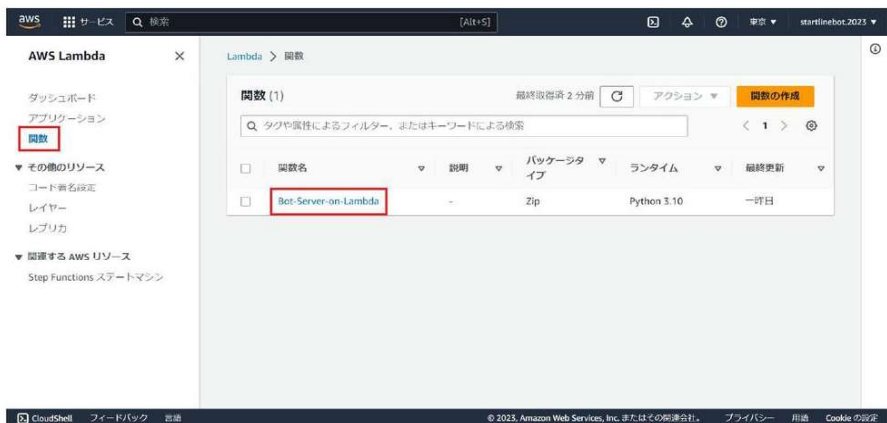


▲図 2.118 レイヤーができた！

次は、作成した OpenAI API SDK のレイヤーを Lambda 関数から利用するための設定を行います。左メニューから「関数」を開いてください。

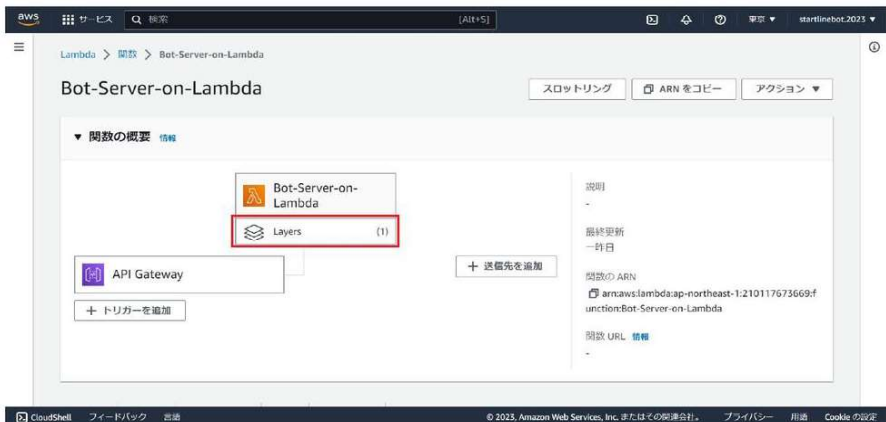
Lambda 関数にレイヤーを追加する

Lambda 関数の一覧で、「Bot-Server-on-Lambda」をクリックします。（図 2.119）



▲図 2.119 「Bot-Server-on-Lambda」をクリックする

この Lambda 関数に、先に作っておいた Messaging API SDK のレイヤーを追加したいので、[Layers] をクリックします。(図 2.120)



▲図 2.120 [Layers] をクリックする

[レイヤーの追加] をクリックします。(図 2.121)



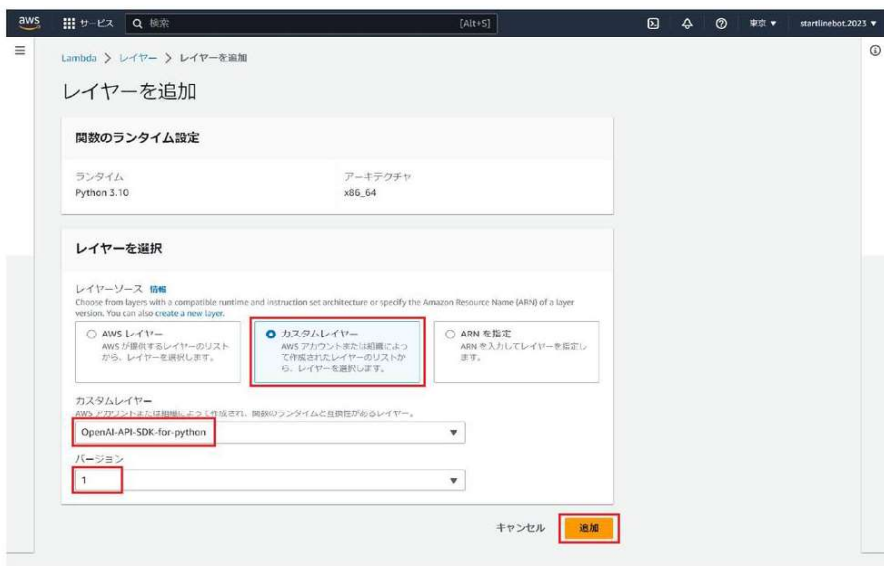
▲図 2.121 [レイヤーの追加] をクリックする

レイヤーは次のように選択します。(表 2.7)

▼表 2.7 レイヤーを選択

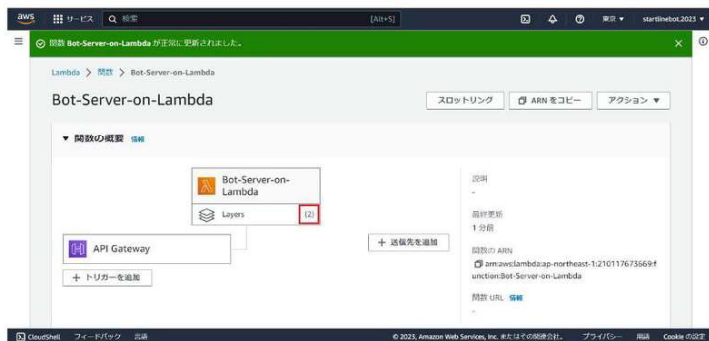
レイヤーソース	カスタムレイヤー
カスタムレイヤー	OpenAI-API-SDK-for-python
バージョン	1

使用するレイヤーを選択したら「追加」をクリックします。(図 2.122)



▲図 2.122 「追加」をクリックする

[Layers] の後ろの数字が (1) から (2) になりました。これで Lambda 関数に OpenAI API SDK のレイヤーが追加できました！ こうしてレイヤーを追加することで、Lambda 関数から OpenAI API SDK が使えるようになります。(図 2.123)



▲図 2.123 OpenAI API SDK のレイヤーが追加できた！

2.7.6 AWS Lambda のコードに「OpenAI の API で質問の回答を取得する処理」を追加する

OpenAI SDK を使う準備ができたので、ユーザーの質問に対して AI チャットボットが自動で応答するようにコードを変更します。Lambda 関数の [コード] タブのコードを、丸ごと次のコード^{*49}に置き換えてください。(リスト 2.6)

▼リスト 2.6 AI チャットボットが自動で応答するコード

```
1: import json
2: import logging
3: import openai
4: import os
5: import sys
6:
7: from linebot import LineBotApi, WebhookHandler
8: from linebot.exceptions import InvalidSignatureError, LineBotApiError
9: from linebot.models import MessageEvent, TextMessage, TextSendMessage
10:
11: # INFOレベル以上のログメッセージを拾うように設定する
12: logger = logging.getLogger()
13: logger.setLevel(logging.INFO)
14:
15: # 環境変数からMessaging APIのチャネルアクセストークンとチャネルシークレットを取得する
16: CHANNEL_ACCESS_TOKEN = os.getenv('CHANNEL_ACCESS_TOKEN')
17: CHANNEL_SECRET = os.getenv('CHANNEL_SECRET')
18:
19: # 環境変数からOpenAI APIのシークレットキーを取得する
20: openai.api_key = os.getenv('SECRET_KEY')
21:
22: # それぞれ環境変数に登録されていないとエラー
23: if CHANNEL_ACCESS_TOKEN is None:
24:     logger.error(
25:         'LINE_CHANNEL_ACCESS_TOKEN is not defined as environmental variables.')
26:     sys.exit(1)
27: if CHANNEL_SECRET is None:
28:     logger.error(
29:         'LINE_CHANNEL_SECRET is not defined as environmental variables.')
30:     sys.exit(1)
31: if openai.api_key is None:
32:     logger.error(
33:         'Open API key is not defined as environmental variables.')
34:     sys.exit(1)
35:
36: line_bot_api = LineBotApi(CHANNEL_ACCESS_TOKEN)
37: webhook_handler = WebhookHandler(CHANNEL_SECRET)
```

^{*49} このコードは GitHub にある本書のリポジトリで公開されています。PDF から直接コピー＆ペーストすると行番号などが混入してしまうため、ぜひこちらを使ってください。 <https://github.com/mochikoAsTech/startLINEBot/blob/master/articles/aichatbot.py>

```

38:
39: # 質問に回答をする処理
40:
41:
42: @webhook_handler.add(MessageEvent, message=TextMessage)
43: def handle_message(event):
44:
45:     # ChatGPTに質問を投げて回答を取得する
46:     question = event.message.text
47:     answer_response = openai.ChatCompletion.create(
48:         model='gpt-3.5-turbo',
49:         messages=[
50:             {'role': 'user', 'content': question},
51:         ],
52:         stop=['。 ']
53:     )
54:     answer = answer_response["choices"][0]["message"]["content"]
55:     # 受け取った回答のJSONを目視確認できるようにINFOでログに吐く
56:     logger.info(answer)
57:
58:     # 応答トークンを使って回答を応答メッセージで送る
59:     line_bot_api.reply_message(
60:         event.reply_token, TextSendMessage(text=answer))
61:
62: # LINE Messaging APIからのWebhookを処理する
63:
64:
65: def lambda_handler(event, context):
66:
67:     # リクエストヘッダーにx-line-signatureがあることを確認
68:     if 'x-line-signature' in event['headers']:
69:         signature = event['headers']['x-line-signature']
70:
71:     body = event['body']
72:     # 受け取ったWebhookのJSONを目視確認できるようにINFOでログに吐く
73:     logger.info(body)
74:
75:     try:
76:         webhook_handler.handle(body, signature)
77:     except InvalidSignatureError:
78:         # 署名を検証した結果、飛んできたのがLINEプラットフォームからのWebhookでな
79:         # れば400を返す
80:         return {
81:             'statusCode': 400,
82:             'body': json.dumps('Only webhooks from the LINE Platform will
83: be accepted.')
84:         }
85:     except LineBotApiError as e:
86:         # 応答メッセージを送ろうとしたがLINEプラットフォームからエラーが返ってきたら
87:         # エラーを吐く
88:         logger.error('Got exception from LINE Messaging API: %s\n' % e.message)
89:         for m in e.error.details:
90:             logger.error(' %s: %s' % (m.property, m.message))
91:
92:     return {
93:         'statusCode': 200,
94:         'body': json.dumps('Hello from Lambda!')}

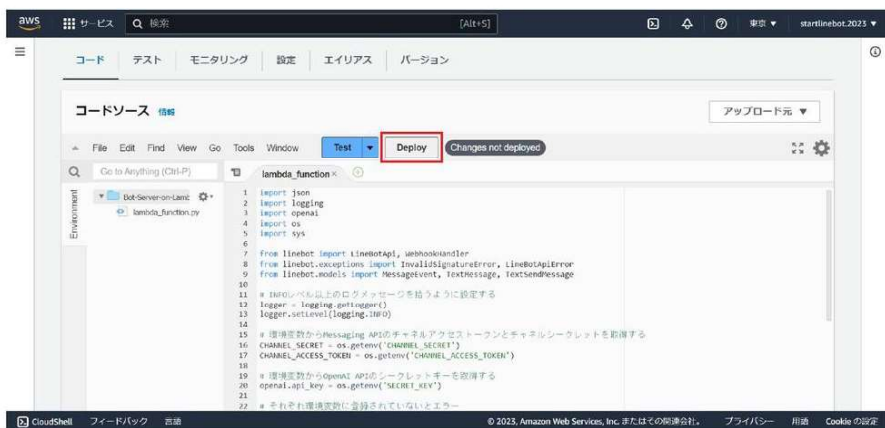
```

このコードの中で、次の部分が OpenAI API をたたいて質問の回答を取得するコードです。OpenAI API に質問を投げると、回答の生成に非常に時間がかかるため、`stop='。']`を指定することで、「最初に句点（。）が出てきたらそこで生成を終了する」というリクエストにして回答生成を早めに切り上げるようにしています。（リスト 2.7）

▼リスト 2.7 OpenAI API をたたいて質問の回答を取得する部分のコード

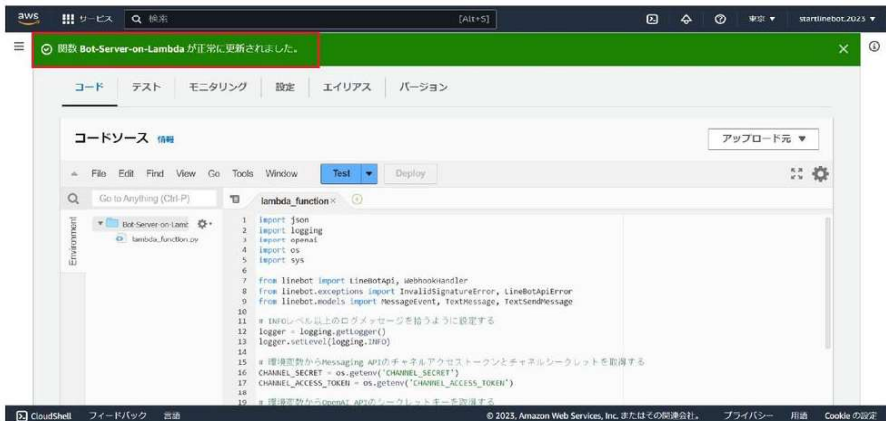
```
45: # ChatGPTに質問を投げて回答を取得する
46: question = event.message.text
47: answer_response = openai.ChatCompletion.create(
48:     model='gpt-3.5-turbo',
49:     messages=[
50:         {'role': 'user', 'content': question},
51:     ],
52:     stop=['。']
53: )
54: answer = answer_response["choices"][0]["message"]["content"]
```

コードを直したら、[Deploy] を押してデプロイ（変更後のコードを反映）します。（図 2.124）



▲図 2.124 [Deploy] を押して変更後のコードを反映する

「関数 Bot-Server-on-Lambda が正常に更新されました。」と表示されたらデプロイ完了です。(図 2.125)



▲図 2.125 デプロイ完了

2.7.7 Lambda 関数のタイムアウトまでの時間を延ばす

いまデプロイした AI チャットボットのコード（リスト 2.6）で、45 行目以降の OpenAI API に質問を投げて回答を取得する部分は、回答が返ってくるまでに数秒以上かかる場合があります。^{*50}

実は Lambda 関数には、コードを実行してから n 秒立ったらタイムアウトする、つまり処理を打ち切る、という最長実行時間があります。[設定] タブの [一般設定] を開くと、[タイムアウト] が **0分3秒** になっていますね。タイムアウトのデフォルト値はこの通り 3 秒なので、Webhook が届いてコードを実行しはじめてから 3 秒以内にすべての処理が終わらないと、そこで処理が打ち切られてしまうのです。[編集] をクリックしてください。（図 2.126）

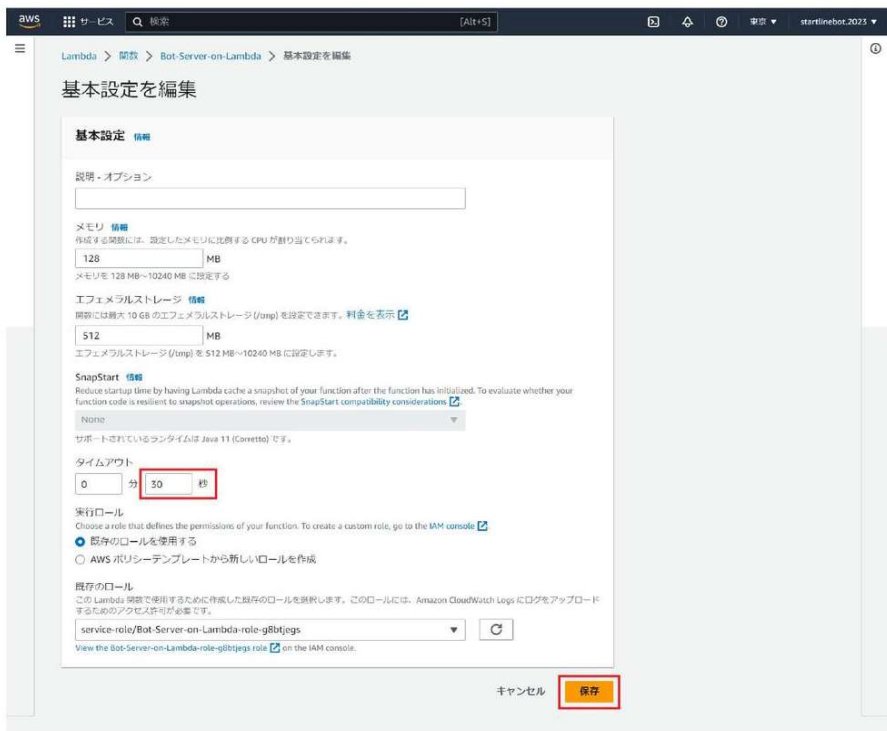


▲ 図 2.126 [編集] をクリックしてタイムアウトの秒数を変更しよう

このままの設定だと、Webhook を届いて OpenAI API に質問を投げてから回答が返ってくるまでに 3 秒以上かかった場合、タイムアウトして処理が打ち切られてしまうため、この [タイムアウト] の秒数を **3秒から 30秒**^{*51}に変更して [保存] します。（図 2.127）

^{*50} 「2.7.3 OpenAI API に質問を投げて回答を取得する」で curl コマンドをたたいたとき、レスポンスが返ってくるまでにちょっと待ち時間があったことを思い出してください。

^{*51} ちなみに AWS Lambda の手前には API Gateway にもタイムアウトがあり、そちらは最大 29 秒で変更もできません。そのため本書の構成だと、仮に Lambda 関数のタイムアウトをもっと長い 60 秒や 90 秒にしたとしても、先に API Gateway がタイムアウトしてしまうのであまり意味がありません。



▲図 2.127 「タイムアウト」を 30 秒に変更して「保存」しよう

Lambda 関数の「タイムアウト」が 30 秒に変更されました。(図 2.128)



▲図 2.128 「タイムアウト」が 30 秒になった

それでは最後に、環境変数の設定を追加しましょう。

2.7.8 環境変数に OpenAI のシークレットキーを追加する

いまデプロイした AI チャットボットのコードを実際に動かすには、OpenAI API のシークレットキーが必要です。チャンネルアクセストークンやチャンネルシークレットと同様に、シークレットキーもソースコードには直接書かず、環境変数として設定しておいて、コードからは `openai.api_key = os.getenv('SECRET_KEY')` というように環境変数を参照する形にしています。

Lambda 関数の「設定」タブから「環境変数」を開いて、「編集」をクリックします。（図 2.129）



▲図 2.129 「環境変数」の「編集」をクリックする

「環境変数の追加」をクリックして、「キー」と「値」を次のように設定します。シークレットキーは、「2.7.2 OpenAI に登録してシークレットキーを取得する」でコピーしてメモ帳に保存してあるはずです。（表 2.8）

▼表 2.8 環境変数の編集

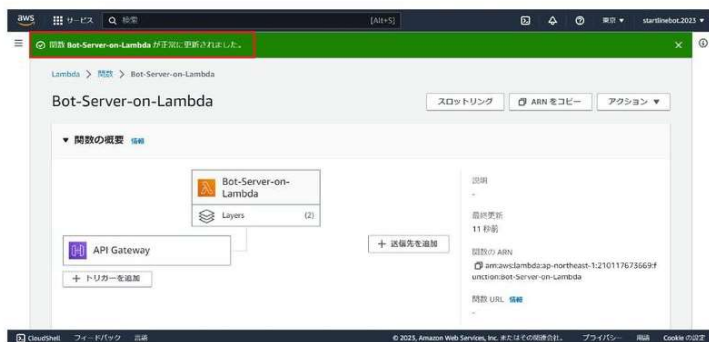
キー	値
SECRET_KEY	シークレットキー

環境変数を編集したら「保存」をクリックします。(図 2.130)



▲図 2.130 環境変数を編集したら「保存」をクリックする

「関数 Bot-Server-on-Lambda が正常に更新されました。」と表示されたら、環境変数に OpenAI のシークレットキーを追加する設定は完了です。(図 2.131)

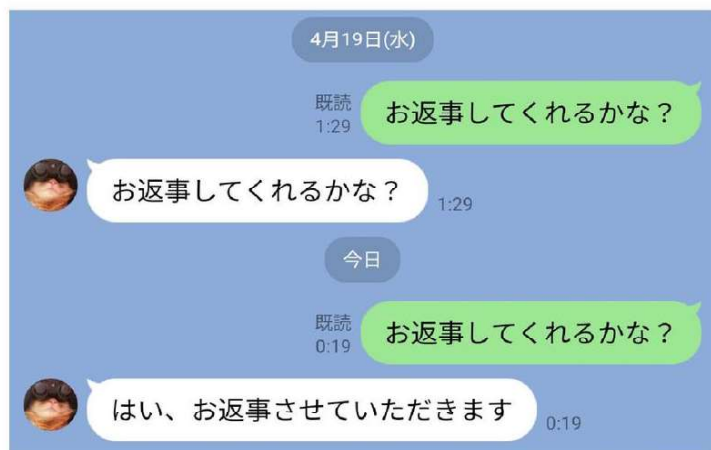


▲図 2.131 環境変数の設定完了

これでボットサーバーのボットを、オウム返しするボットから AI チャットボットに切り替える作業が終わりました。

2.7.9 LINE 公式アカウントに話しかけて AI チャットボットの回答を確認しよう

それでは LINE 公式アカウントに質問をしてみて、AI チャットボットが回答してくれることを確認してみましょう。(図 2.132)



▲図 2.132 メッセージを送ったら AI チャットボットが返事してくれた

さっきはメッセージをオウム返しをするだけだったのに、AI チャットボットが回答をしてくれるようになりました！ おめでとうございます！ これで「AI チャットボットをつくる」というアチーブメントも達成です！

【コラム】Webhook へのレスポンスが先？ 応答メッセージが先？

ボットサーバーで Webhook を受け取った際、Webhook のその内容に応じた処理や応答メッセージの送信などをしてからステータスコード 200 を返す、という順番はお勧めしません。

なぜなら間に挟まる処理や応答メッセージ送信などに時間がかかった場合、レスポンスを返す前に LINE プラットフォームから送られた Webhook のリクエストがタイムアウトしてしまう可能性があるためです。

なんとなく先に色々片付けてからレスポンスを返したい気持ちは分かりますが、郵便局員が郵便物を持って来てくれたとき、わざわざ玄関前で待たせて封筒を開けて中身を読んでからようやく受領の判子を押してあげる、という順番だとおかしいよね、と考えると納得しやすいのではないのでしょうか。まずは受領の判子を押して郵便局員を帰らせてあげて、手紙を読んだり、返事を書いたりはそのあとでやりましょう。^{*52}

またメッセージに対する応答メッセージを送る場合、応答トークン^{*53}は Webhook を受信後、速やか（具体的には 1 分以内）に使う必要があります。Webhook 受信後、すぐに返信を送れる場合は応答メッセージで構いませんが、色々な処理を行ってかなり時間が経ってから返信したい場合は、応答トークンがすでに無効になっている可能性があります。その場合は、対象となる友だちのユーザー ID を指定したプッシュメッセージ^{*54}で返信を送ることが可能です。

^{*52} この「Webhook を受け取ったら、先にレスポンスを返してその後に応答メッセージなどの処理をすべき」という説明を読んで、勘の良い読者の方はもうお気づきかもしれませんが、実は本書ではオウム返しボットのコードも AI チャットボットのコードも、郵便局員を待たせまくって、なんなら返信を書いてポストに投函してからようやく受領の判子を押しています。非同期処理まで含めると手順や説明が非常に複雑になってしまうため、本来はあまり推奨されない方法だと理解した上で、今回は動くものを簡単に作れることを優先してこのようにしています。

^{*53} 応答トークンはすぐ消費期限の短い「メッセージ無料送信チケット」のようなものです。応答トークンは、LINE 公式アカウントが友だち追加されたときや、LINE 公式アカウントにメッセージが送られたときに飛んでくる Webhook に含まれていて、友だち追加やメッセージに対する返信を送るときに使えます。 <https://developers.line.biz/ja/reference/messaging-api/#send-reply-message-reply-token>

^{*54} プッシュメッセージを送る | Messaging API リファレンス | LINE Developers <https://developers.line.biz/ja/reference/messaging-api/#send-push-message>

CHAPTER 3

Messaging APIの 色々な機能を試してみよう

色々な機能を組み合わせて自分だけのLINE Botをつくろう！

3.1 メッセージ送信に関する機能

LINE 公式アカウントからメッセージを送るとき、普通に送るだけでなく、特定の人や属性を指定して送ったり、見た目にこだわったメッセージを送ったりすることができ
ます。

3.1.1 ユーザー ID を指定して特定の人に送る

Messaging API でメッセージを送るとき、いちばん簡単なのは友だち全員にメッセージを一斉配信するブロードキャストメッセージです。ですがユーザー ID を指定して、特定の人にだけメッセージを送ることも可能です。特定のひとりにだけ送りたいときはプッシュメッセージ、特定の数人にまとめて送りたいときはマルチキャストメッセージで送れます。

友だちのユーザー ID は、LINE プラットフォームから飛んでくる Webhook の `source` オブジェクトに含まれています。^{*1}

【コラム】開発チームだけにメッセージのテスト配信がしたい

たとえば本番用の LINE 公式アカウントと、テスト用の LINE 公式アカウントを別々に用意しておいて、開発チームのメンバーだけがテスト用の LINE 公式アカウントと友だちになることで、本番配信前のメッセージのテスト配信と確認を実現していたとします。

この場合、開発チームに所属していたメンバーが A さん、B さん、C さんの中で C さんが退職してしまうと、残った A さん、B さん側で C さんをテスト用の LINE 公式アカウントの友だちからブロックする方法がありません。退職前に C さんに頼んで、C さん側からテスト用の LINE 公式アカウントをブロックしてもらえないのです。

さらに厳密に言うと、一度 A さん、B さんの目の前で C さんにブロックの操作をしてもらったとしても、C さんの操作でブロックは解除できるので、悪意のある C さんが後日ブロックを解除してしまうことはとめられません。こうなると、

^{*1} ユーザー ID を取得する | LINE Developers <https://developers.line.biz/ja/docs/messaging-api/getting-user-ids/>

退職した C さんにも、発売前の商品や未発表の情報を含むテスト配信のメッセージが届き続けてしまいます。

なのでメッセージのテスト配信の仕組みは、「一度友だち追加されたら、開発者側からは友だち状態はコントロールできない」という前提で組んでおく必要があります。

たとえば、テスト配信のメッセージはテスト用の LINE 公式アカウントから送るとしても、ブロードキャストメッセージで友だち全員に送るのではなく、開発チームの A さん、B さん、C さんのユーザー ID を指定したマルチキャストメッセージを送る、という方法がおすすめです。これなら C さんの退職後、開発チームに残った A さん B さんがマルチキャストメッセージの送信対象から C さんのユーザー ID を消せば、C さんには公開前のテストメッセージは届かなくなります。

3.1.2 メッセージの配信対象を属性で絞り込む

ナローキャストメッセージ^{*2}を使うと、性別や年齢、地域、友だちになってからの期間といった属性情報を指定してメッセージを送ることが可能です。ただし属性情報を指定したメッセージ送信には、LINE 公式アカウントのターゲットリーチが 100 人以上で、かつ属性情報を指定して絞り込んだ送信対象も 50 人以上であること、という制限事項があります。そもそも友だちが少なかったり、条件を絞り込みすぎて送信対象者が少なかったりすると、属性情報を指定したメッセージ送信はできませんので注意してください。^{*3}

たとえばブロードキャストメッセージで新商品予告のメッセージを全員に送って、そのメッセージの URL を開いたユーザーを「新商品に興味のあるオーディエンス」に入れておき、新商品の発売当日はそのオーディエンスだけを対象にして店頭イベントの告知を送る、というようなこともできます。

^{*2} 属性情報やリターゲティングを利用して複数のユーザーに送信する（ナローキャストメッセージ） | LINE Developers <https://developers.line.biz/ja/docs/messaging-api/sending-messages/#send-narrowcast-message>

^{*3} 属性情報やオーディエンスを利用したメッセージ送信の制限事項 | LINE Developers <https://developers.line.biz/ja/reference/messaging-api/#send-narrowcast-message-restrictions>

3.1.3 メッセージ送信元のアイコンと表示名を変更する

メッセージを送るときに、送信元のアイコンと表示名を変更して送ることができます。^{*4}

次の curl コマンド^{*5}（リスト 3.1）の 3 行目にある{チャンネルアクセストークン}の部分
を、Messaging API チャンネルのチャンネルアクセストークンに置き換えてください。チャ
ネルアクセストークンは、先ほど「2.4.1 LINE Developers コンソールでチャンネルアクセ
ストークンを発行する」でコピーしましたね。

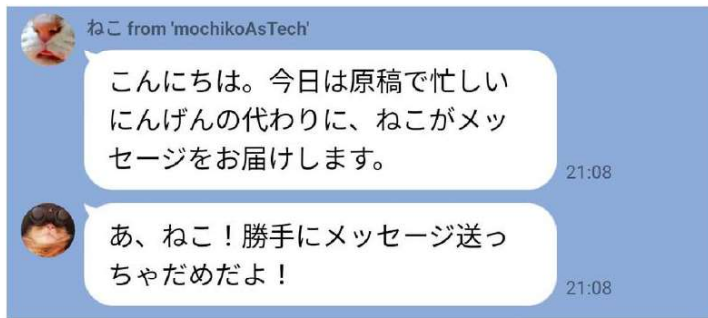
WSL またはターミナルを起動して curl コマンドをたたくと、LINE 公式アカウントか
らこんなメッセージが届きます。（図 3.1）

▼リスト 3.1 アイコンや表示名を変更してメッセージを送る curl コマンド

```
1: curl -v -X POST https://api.line.me/v2/bot/message/broadcast \
2: -H 'Content-Type: application/json' \
3: -H 'Authorization: Bearer {チャンネルアクセストークン}' \
4: -d '{
5:     "messages": [
6:         {
7:             "type": "text",
8:             "text": "こんにちは。今日は原稿で忙しいにんげんの代わりに、ねこがメッ
9: セージをお届けします。",
10:            "sender": {
11:                "name": "ねこ",
12:                "iconUrl": "https://pbs.twimg.com/media/FueHfQNaYAEgNUj?format
13: =jpg&name=900x900"
14:            },
15:            {
16:                "type": "text",
17:                "text": "あ、ねこ！勝手にメッセージ送っちゃだめだよ！"
18:            }
19:        ]
20:    }'
```

^{*4} アイコンおよび表示名を変更する | LINE Developers <https://developers.line.biz/ja/docs/messaging-api/icon-nickname-switch/>

^{*5} このコードは GitHub にある本書のリポジトリで公開されています。PDF から直接コピー＆ペーストすると行番号などが混入してしまうため、ぜひこちらを使ってください。 <https://github.com/mochikoAsTech/startLINEBot/blob/master/articles/change-icon-and-send-message.sh>



▲図 3.1 ねこからメッセージが届いた

たとえばテーマパークの LINE 公式アカウントで、特定のキャラクターにちなんだイベントを告知するときだけ、メッセージの送信元をそのキャラクターのアイコンと名前にする、といった使い方が可能です。

【コラム】 URL を送る前に OGP の見た目を確認したり、キャッシュを消したりしたい

LINE で URL を含むメッセージを送ると、こんなふうにプレビューが表示されます。(図 3.2)



▲図 3.2 URL のプレビュー

実際にメッセージを送る前に、このプレビューでどんな画像やテキストが表示されるのか、確認したかったらどうすればいいのでしょうか？

PagePoker という公式のツールを使うと、対象ページの OGP タグ^{*6}を読み込んで、どんなふうにプレビューが表示されるのかを事前に確認できます。^{*7}

- PagePoker
 - <https://poker.line.naver.jp/>

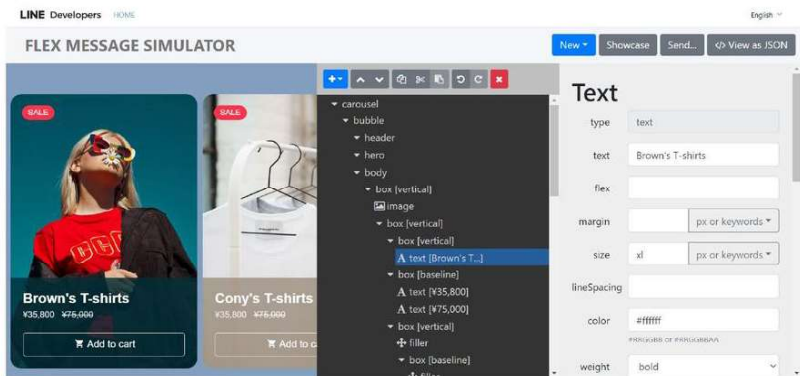
[Clear Cache] にチェックを入れることで、文字通り LINE 側サーバーのキャッシュをクリアできます。ページの OGP 画像を差し替えたなら、ここで先にキャッシュをクリアしてからそのページの URL を含むメッセージを送るようにすることで、「OGP 画像を差し替えたのになぜか古い画像がプレビューで表示されてしまった」という事態は避けられそうです。OGP タグの書き方については、LINE Developers サイトの FAQ^{*8}を参考にしてください。

3.1.4 Flex Message で見た目にこだわったメッセージを送る

単体のテキストや画像ではなく、もっと格好良い、見た目にこだわったメッセージを送りたい！と思ったら、Flex Message Simulator を使ってみましょう。

- Flex Message Simulator^{*9}
 - <https://developers.line.biz/flex-simulator/>

GUI でレイアウトをカスタマイズするだけで、メッセージの JSON^{*10}が生成される素晴らしいツールです。(図 3.3)



▲図 3.3 Flex Message Simulator で見た目にこだわったメッセージをつくってみよう

^{*6} OGP は Open Graph Protocol の略です。HTML にメタデータとして「og:image」のような OGP タグを書いておくことで、Twitter や LINE などでもその URL を共有したときに、URL そのままではなく対象ページのタイトルや概要、画像などがカードのように表示されます。

^{*7} Twitter の Card Validator (<https://cards-dev.twitter.com/validator>) とか、Facebook のシェアデバッガー (<https://developers.facebook.com/tools/debug/>) みたいなものですね。Twitter の Card Validator は気づいたらプレビュー確認機能がなくなっていました。

^{*8} トークと LINE VOOM 内の URL プレビューはどのようにして生成されますか？ | LINE Developers <https://developers.line.biz/ja/faq/#how-are-the-url-previews-generated>

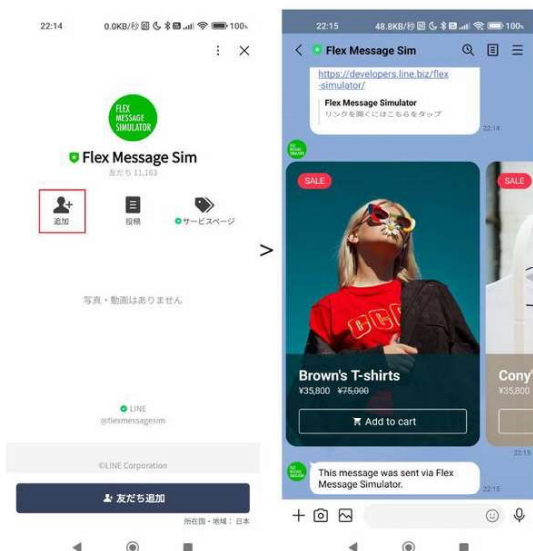
^{*9} Flex Message Simulator を使うときは、LINE Developers コンソールと同じように LINE アカウントでログインする必要があります。

^{*10} ただし Flex Message Simulator で出力される JSON は、messages 直下の Flex Message 全体ではなく、コンテナ (contents 以下) なので注意が必要です。Flex Message の仕様については、API リファレンスを参照してください。 <https://developers.line.biz/ja/reference/messaging-api/#flex-message>

しかも「この Flex Message を送ったら実際はどんな見た目になるんだろう？ 画像は見切れないかな？」と思ったら、右上の [Send...] からメッセージをテスト送信できるので。便利！（図 3.4、図 3.5）



▲図 3.4 Flex Message Simulator からテスト送信する



▲図 3.5 Flex Message Simulator の LINE 公式アカウントからメッセージが届く

3.1.5 ボットサーバーが Webhook を受け取れなかったときの再送機能

たとえば LINE 公式アカウントが急に有名になり、一気に友だちが増えて、メッセージが大量に送られてきたとします。急なアクセス集中でサイトが落ちるように、LINE プラットフォームから飛んでくる Webhook のリクエストによって、ボットサーバーが過負荷になって応答できなくなってしまうらどうなるのでしょうか。

そんなときのために「Webhook の再送^{*11}」という機能があります。Webhook の再送はデフォルトではオフになっていますが、LINE Developers コンソールでオンにすることで、ボットサーバーが Webhook を受け取れなかったときに再送してくれるようになります。

どこで「Webhook を受け取れなかった」と判断されるのかというと、LINE プラットフォームから見て、「Webhook を受け取ってー！」というリクエストに対して、ボットサーバーからステータスコード 200 が返ってこなければ「ボットサーバーは Webhook を受け取れなかった」と判断されます。

とても便利に見える Webhook の再送機能ですが、ボットサーバーから見て「Webhook を受け取ってー！」というリクエストに対して、きちんとステータスコード 200 を返したつもりでも、ボットサーバーから LINE プラットフォームまでのネットワーク経路上で何か問題があって、きちんとレスポンスが LINE プラットフォームまで到達しないケースは考えられます。その場合、Webhook の再送をオンにしたことで次のようなトラブルが考えられます。

1. ユーザーが『購入する』というメッセージを送信する
2. LINE プラットフォームからボットサーバーに『購入する』というメッセージの Webhook が飛ぶ
3. ボットサーバーで Webhook を受け取ってステータスコード 200 を返し、商品購入処理を行った上でユーザーに「購入完了しました！」という応答メッセージを送る
4. しかし何らかの理由でステータスコード 200 のレスポンスが LINE プラットフォームに到達しなかった
5. LINE プラットフォームは「ボットサーバーが Webhook を受け取れなかった」と判断して、『購入する』というメッセージの Webhook を再送する
6. ボットサーバーで Webhook を受け取ってステータスコード 200 を返し、商品購入処理を行った上でユーザーに「購入完了しました！」という応答メッセージを送る

^{*11} 受け取りに失敗した Webhook を再送する | LINE Developers <https://developers.line.biz/ja/docs/messaging-api/receiving-messages/#webhook-redelivery>

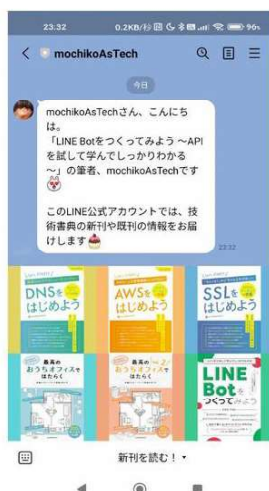
7. ユーザーは一度『購入する』と送っただけなのに、商品が2つ購入されてしまい、購入完了メッセージも2度届いて驚く

これを防ぐためには、ボットサーバー側で商品購入処理を行う際に、Webhook のイベントごとに一意な値である `webhookEventId` を確認して、「すでに購入処理を行った Webhook イベントと同一のイベントではないか？」を確認する必要があります。

Webhook の再送は便利な機能ですが、このように意図しない再送が行われたときのことを考えた重複チェックの処理がなければ、迂闊にオンにするべきではありません。

3.2 リッチメニュー

LINE 公式アカウントと友だちになると、トーク画面の下部にこんなメニューが表示されることがありますよね。これは簡素なテキストではなく、画像でできた贅沢で多彩なメニュー…つまりリッチなメニュー…リッチなメニューなので…リッチメニュー！ そう、これがリッチメニュー^{*12}です。(図 3.6)



▲図 3.6 リッチメニュー

^{*12} リッチメニューを使う | LINE Developers <https://developers.line.biz/ja/docs/messaging-api/using-rich-menus/>

このリッチメニューは、一番下の「新刊を読む! ▼」と書いてあるバーをタップすることで開いたり閉じたりできます。このバーのテキストも、デフォルトの「メニュー」から「新刊を読む!」「メニューはこちら」「お問い合わせの入力ははこちら」「ほっとするブレイクタイムを」「会員証」「お役立ち情報ははこちら」「人気ランキングをチェック!」「こちらをチェック」「荷物の追跡・再配達・集荷受付」といったさまざまなテキストに変更できます。

リッチメニューの実態は次のような1枚の画像であり、画像内の各領域にそれぞれリンクを設定することで、公式サイトや予約ページ、特定の機能などにユーザーを誘導できます。(図3.7)



▲図 3.7 リッチメニューの画像

3.2.1 リッチメニュープレイグラウンドでリッチメニューを体験してみよう

リッチメニューのさまざまな機能は、文字で説明するより体験してみるのが一番分かりやすいです。公式で提供されているリッチメニュープレイグラウンド^{*13}という、「リッチメニューを体験するための LINE 公式アカウント」と友だちになってみましょう。(図3.8)

^{*13} リッチメニュープレイグラウンドと友だちになる。 <https://lin.ee/7ALASDvA>



▲図 3.8 リッチメニュープレイグラウンドと友だちになる

「リッチメニューを開く▼」をタップすると、「メッセージアクションを試す」と表示されました。これは「ユーザーがリッチメニューをタップすることで、特定のメッセージをユーザーから自動送信させる」という機能です。早速「メッセージ送信」を試してみましょう。(図 3.9)



▲図 3.9 「メッセージ送信」をタップするとメッセージが送信されて返信が届いた

あなたが文字を入力した訳ではないのに、「message sent successfully!」というメッセージがあなたから送信されました。これがメッセージアクションです。メッセージアクションによって、ユーザーから LINE 公式アカウントに対して自動でメッセージが送られ、それによって Webhook がボットサーバーに届きました、という一連の流れを解説してくれています。

たとえば「再配達申し込みをしたかったら、LINE で『再配達』というメッセージを送ってね」のように、ユーザーに手入力を促さなくても、リッチメニューに「再配達申し込み」というボタンを用意して、そのボタンの領域をタップしたら「再配達」というメッセージが自動送信されるよう、対象の領域にメッセージアクションを設定しておけばいいのです。このように「この機能はこんな風に使えばいいのか」を体験できるのがリッチメニュープレイグラウンドのいいところです。

3.2.2 リッチメニューの画像を作る

リッチメニューの画像を作ろうと思うと意外と大変です。LINE Official Account Manager でリッチメニューの作成画面を開くと、「デザインガイド」というボタンがあり、そこからリッチメニューのテンプレート画像がダウンロードできます。Messaging API を使ったリッチメニューの設定を試したい場合は、このテンプレートをそのまま使ってみるのがお勧めです。(図 3.10)



▲図 3.10 リッチメニューのテンプレート画像がダウンロードできる

3.2.3 LINE Official Account Manager と Messaging API でのリッチメニューの制約の違い

リッチメニューは Messaging API で頑張って JSON を組み立てて作らなくても、実は LINE Official Account Manager で画像やテキストを組み合わせて作ることも可能です。

ただし LINE Official Account Manager でリッチメニューを作る場合、大なら7種類、小なら5種類ある特定のサイズや形からしか選べません。また全員に同じリッチメニューを出す機能しかないため、会員証を持っている人にだけ会員向けのリッチメニューを出す、といった出し分けもできません。Messaging API であれば、縦横のサイズや、それぞれの領域も自由に設定でき、ユーザーごとに別々のリッチメニューを表示させることも可能ですし、タブ切り替えのようなリッチメニューを作ることもできます。

限定された機能で構わないのでラクにリッチメニューが設定したいなら LINE Official Account Manager を使って、こだわったリッチメニューが設定したいなら Messaging API を使うのが良いでしょう。

3.3 Messaging API をもっと楽しむために

ここまで Messaging API を使って LINE Bot を作ってきましたがいかがでしたか？楽しんでもらえましたか？

最後に、一步踏み出したあなたがこれから Messaging API をもっと楽しむために、「この先にはこんな道やあんな道がありますよ」という、さまざまな道のご紹介をしたいと思います。

3.3.1 Messaging API 以外のプロダクトとの組み合わせ

LINE API には、Messaging API の他に、LINE ログインや LIFP (LINE Front-end Framework) や LINE ミニアプリ、LINE Social Plugins などさまざまなプロダクトがあります。Messaging API 単体でできることが把握できたら、他のプロダクトと組み合わせるとさらにどんなことができるのか、LINE Developers サイトでドキュメントを見てみましょう。

- LINE Developers サイト
 - <https://developers.line.biz/>

3.3.2 LINE キャンパスで資格を取ってみよう

「Messaging API の前提知識として、もう少し LINE 公式アカウントについて学びたい」という場合は総合学習プラットフォーム「LINE キャンパス」がお勧めです。「LINE 公式アカウント Basic」と「LINE 公式アカウント Advanced」という認定資格の取得を目指して、学習コースや資格認定コースを無料で受講してみましょう。

- LINE キャンパス
 - <https://campus.line.biz/>

3.3.3 LINE Creative Lab を使おう

LINE Creative Lab を使うと、画像メッセージとして送れる素材が簡単に作れます。クリエイティビティにあふれた画像や動画を作って、Messaging API で送ってみましょう。

- LINE Creative Lab
 - <https://creativelab.line.biz/>

3.3.4 パソコン版の LINE を使おう

意外と知らない人が多いのですが、実はスマートフォンで使っている LINE と同じアカウントで、パソコン版の LINE も使うことができます。Messaging API の動作検証をするため、LINE 公式アカウントにメッセージを送る場合は、PC 版の LINE で作業した方がやりやすいです。ただしリッチメニューなど、パソコン版の LINE ではサポートされていない機能も一部ありますので注意してください。

- パソコンで LINE を利用する | LINE みんなの使い方ガイド
 - <https://guide.line.me/ja/services/pc-line.html>

3.3.5 LINE Developers Community に参加しよう

LINE API を活用している有志によって、LINE Developers Community のイベントが定期的に行われています。LINE API Expert として認定されたみなさんが、最新機能や開発手法をオンラインで教えてくれるので、開発で行き詰まったらひとりで悩まずにコミュニティに参加してみるのがお勧めです。LINE Developers Community のサイト

にはユーザー同士の Q&A もあり、開発でつまづいたときは質問を投稿したり、過去の質問からヒントを探したりできます。

- LINE Developers Community
 - <https://www.line-community.me/>

あとがき

LINE API についての本が書けたらいいな、と最初に思ったのは 2019 年の冬でした。3 年以上経って、念願叶ってこの本を出すに至ったので、やりたいことがあったら、その瞬間は何か理由があって無理でも定期的に機会を窺って「やりたい！」と言い続けるのが大事だな、と思った 2023 年の初夏です。

何かをしようと思うと、そのときやってくるのはだいたい解法の明快な問題などではなく、「これは…どう進めたらいいんだ？ どれも手詰まりで…頑張り方が分からなくてなんかしんどいな…」みたいな胸苦しい障壁ばかりです。進もうとする足を引き留め、膝を折らせようとする何かは、分かりやすい悪意の形はしておらず、一見慕わしい優しさをまとうてやってくることも多いです。あるいは単純に、実力のないものに出番は与えられないというだけの話かもしれません。でもそういうときでも、泣こうが転ぼうが膝を折ろうが最終的に立っていれば勝つので、何度でも立ち上がって走り続けるぞ、という常時ファイティングポーズな気持ちでいます。

2023 年 5 月
mochikoAsTech

PDF 版のダウンロード

本書は下記の URL から PDF 版をどなたでも無料でダウンロードできます。

- ダウンロード URL
 - <https://mochikoastech.booth.pm/items/4717104>

Special Thanks:

- 掛け布団の暖かさに目覚めてしまったねこ
- ディスプレイの明度を 0 にして仕事をさせないねこ
- 目のゴミを取られたくないねこ

レビューアー

- Yuta Kasai
- Kota Momoki
- アリシィ (@shigure_alicey)
- Hiroki Zenigami
- Yuta Kamiyama
- Takuya Kanatani (@torisankanasan)
- Ann Ojes
- Hiroaki Koike
- Kazuyuki Sano

参考文献

- はじめてでもできる！ LINE ビジネス活用公式ガイド - LINE 株式会社
 - <https://book.impress.co.jp/books/1121101033>
- LINE BOT を作ろう！ Messaging API を使ったチャットボットの基礎と利用例 - 立花 翔
 - <https://www.shoeisha.co.jp/book/detail/9784798150734>
- LINE API 実践ガイド - LINE API Expert 認定メンバー
 - <https://book.mynavi.jp/ec/products/detail/id=117310>

- Hands-on LINEBOT - しんぶんぶん
 - <https://techbookfest.org/product/3EUnJ5WbexvCDyMgSJS1qb>
- Real World HTTP - 渋谷 よしき
 - <https://www.oreilly.co.jp/books/9784873118048/>

著者紹介

mochiko / @mochikoAsTech

テクニカルライター。元 Web 制作会社のインフラエンジニア。ねこが好き。「分からない気持ち」に寄り添える技術者になれるように日々奮闘中。技術書典で頒布した「DNSをはじめよう 改訂第2版」「AWSをはじめよう 改訂第2版」「SSLをはじめよう」の「はじめようシリーズ3部作」は累計で11,000冊を突破。

- <https://twitter.com/mochikoAsTech>
- <https://mochikoastech.booth.pm/>
- <https://note.mu/mochikoastech>
- <https://mochikoastech.hatenablog.com/>
- <https://www.amazon.co.jp/mochikoAsTech/e/B087NBL9VM>

Hikaru Wakamatsu

表紙、章扉、目次デザインを担当。

Shinya Nagashio

章扉、目次、挿絵デザインを担当。

LINE Botをつくってみよう

APIを試して学んでしっかりわかる

2023 年 5 月 20 日 技術書典 14 初版

著 者 mochikoAsTech

デザイン Hikaru Wakamatsu / Shinya Nagashio

発行所 mochikoAsTech

(C) 2023 mochikoAsTech