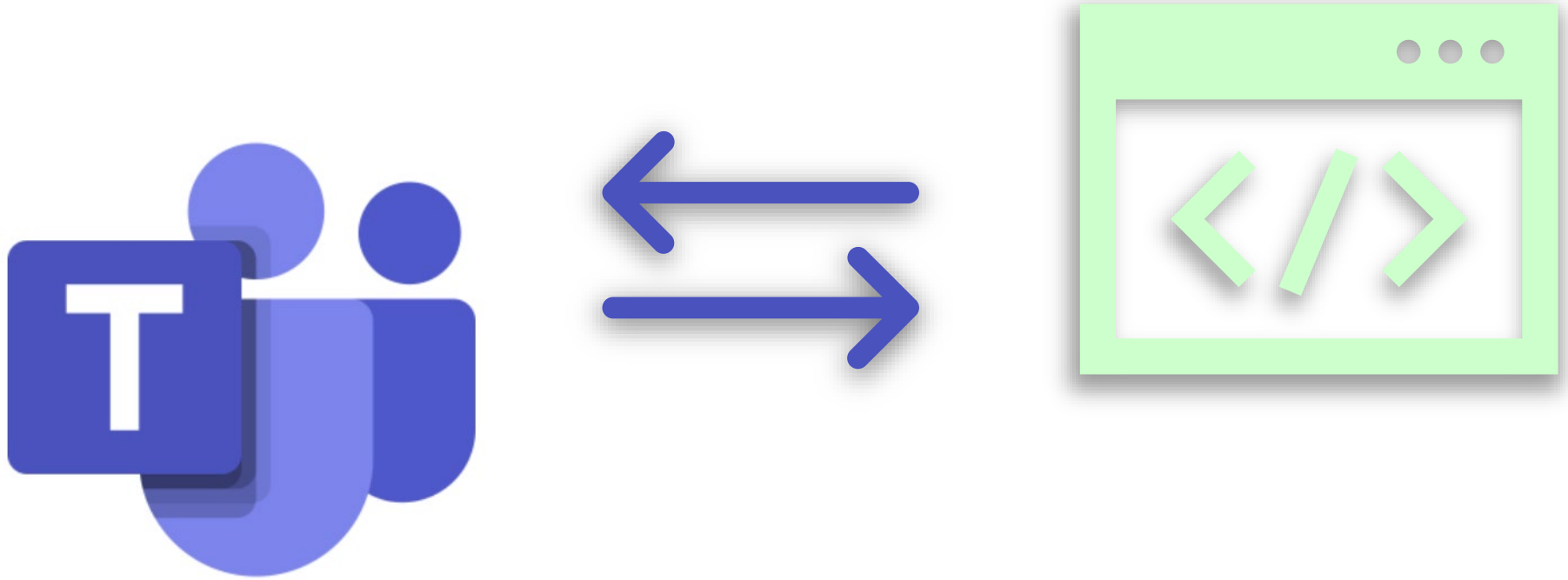
Teamsボットの外部から プロアクティブメッセージを 送信する

外部から、Teamsボットに
プロアクティブメッセージ("通知"、"DM")の
トリガを送るにはどうしたら良いのか?



アダプタ

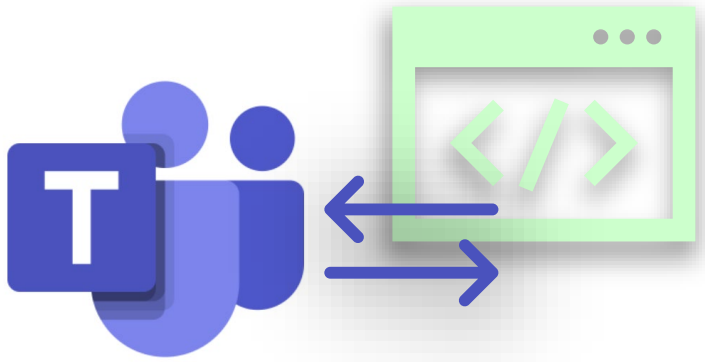
リファレンス

/api/notify

会話

Webフック

POST HTTPS



アダプタ

リファレンス

ダイアログ

アクティビティ

/api/notify

Bot Framework SDK

コネクタ

ターン

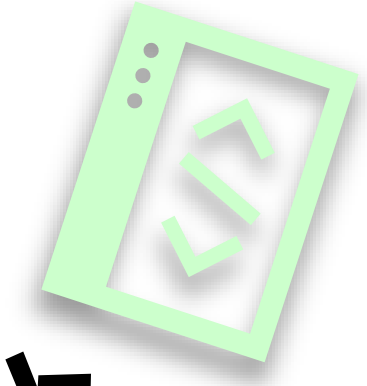
会話 ID

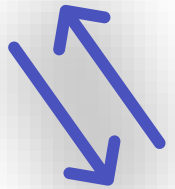
コンテクスト

Webフック

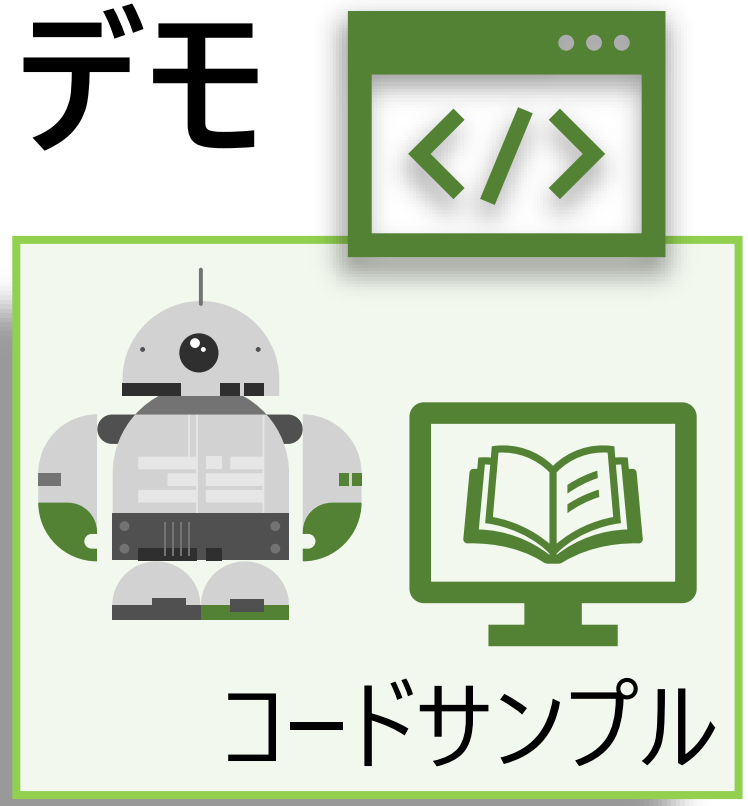
/api/messages

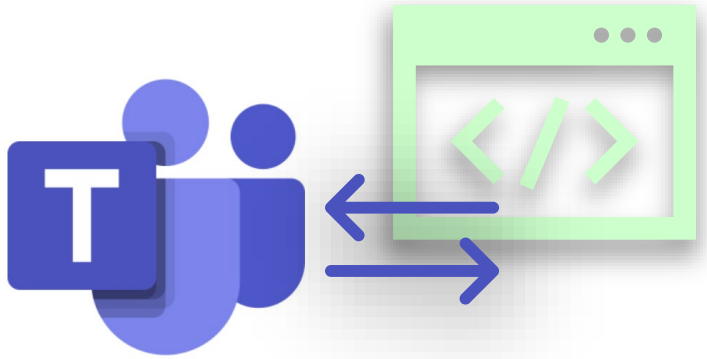
POST HTTPS





デモ





- [x] 実際に動くコードサンプルにより詳細に理解できる
- [x] 順序立てた説明を聞くことにより明瞭に理解できる
- [x] 動画を見るだけなので短時間で理解できる

理解するためのポイント

[ポイント1] 適切なコードサンプルを参照する

[ポイント2]「会話リファレンス」について理解する

[ポイント3] 

について理解する

[ポイント1] 適切なコードサンプルを参照する

△ ① 「/api/notifyを実装しない」方法 (Bot Framework Ver.4)
Send proactive messages - Teams (※JavaScriptのサンプルがその例)

○ ② 「/api/notifyを実装する」方法 (Bot Framework Ver.4)
Send proactive notifications to users - Azure Bot Service

× ③ 「コネクタサービスを直接使用する」方法 (Bot Framework Ver.3)
Proactive messages - Teams

× ④ その他 (Webフック、ノーコード・ローコード、等)

△ 外部からコールできない

① 「/api/notifyを実装しない」方法 (Bot Framework Ver.4)
Send proactive messages - Teams

② 「/api/notifyを実装する」方法 (Bot Framework Ver.4)
Send proactive notifications to users - Azure Bot Service

○ 外部からもコールできる

○

Ver.4ではアダプタ（BotFrameworkAdapter）を使用する

②「/api/notifyを実装する」方法 (Bot Framework Ver.4)
Send proactive notifications to users - Azure Bot Service

③「コネクタサービスを直接使用する」方法 (Bot Framework Ver.3)
Proactive messages - Teams

×

レガシーな手法。
Ver.4ではコネクタサービスを直接使用しない

○ **本格的な実装に使用できる**
（イベントハンドリング、複雑なUI、他との連携、等）

② 「/api/notifyを実装する」方法 (Bot Framework Ver.4)
Send proactive notifications to users - Azure Bot Service

✕ **簡易な実現のみ。本格的な実装には使用できない**

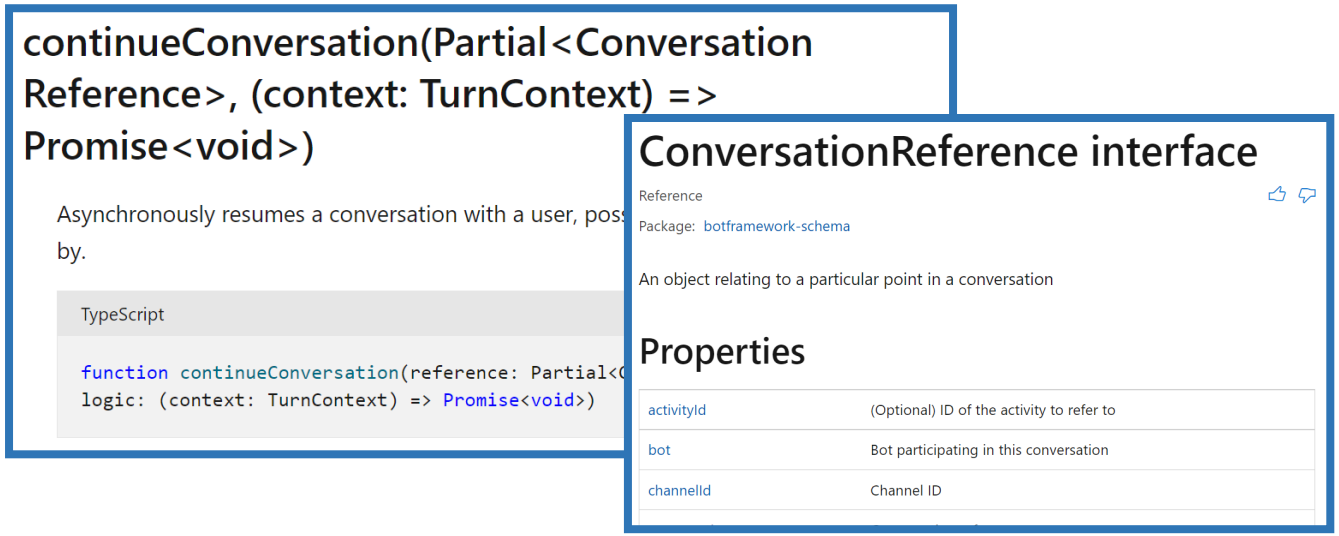
④ その他 (Webフック、ノーコード・ローコード、等)

[ポイント2]「会話リファレンス」について理解する

「**会話リファレンス**」とは

- 「会話」を特定するための識別情報を含む JSON オブジェクト
- 「会話」のある時点の情報を集めた形になっている
- TurnContext.getConversationReference(context.activity)で取得できる
- adapter.**continueConversation** (reference, callback) の第1引数に渡す

プロアクティブメッセージ
を実現するためのメソッド
(非同期的にユーザーとの
「会話」を再開する)



[ConversationReference interface - Azure Bot Service]

continueConversation の第1引数には、「conversationIDとserviceUrlだけ」を渡しても機能するが、「会話リファレンス全体」を渡すのが基本。

「会話リファレンス全体」


```
{
  "activityId": "1649142134564",
  "user": {   "id": "29:1M9wxaUg-r3jWNnZxPuICW5GH [...]",
              "name": "HanakoSato",
              "aadObjectId": "[redacted]"
           },
  "bot": { "id": "28:58cef877-3d88-46ed-b866-fa91a032ca24",
           "name": "APPSSOPROACT4"
         },
  "conversation": { "conversationType": "personal",
         "tenantId": "[redacted]",
         "id": "a:1agvE8HKqkNwgOK3edmH6OZC-Uz [...]"
                   },
  "channelId": "msteams",
  "locale": "ja-JP",
  "serviceUrl": "https://smba.trafficmanager.net/jp/"
}
```


「conversationIDとserviceUrlだけ」


```
{
 "conversation": {
     "id": "a:1agvE8HKqkNwgOK3edmH6OZC-Uz [...]"
                   },
 "serviceUrl": "https://smba.trafficmanager.net/jp/"
}
```

[ポイント3] /api/notify について理解する

[BotFrameworkAdapter class - Azure Bot Service]

「/api/notify」とは

- POSTコールして continueConversation を呼び出すためのエンドポイント
- req.body経由でreference (会話リファレンス) を受け取って continueConversation の第1引数に渡す形になっている

/api/notify エンドポイントにPOSTコールする形

req.bodyからreferenceを受け取っている

referenceをcontinueConversationの第1引数に渡している

```JavaScript
server.post('/api/notifyUser', async (req, res) => {
    // Lookup previously saved conversation reference.
    const reference = await findReference(req.body.refId);

    // Proactively notify the user.
    if (reference) {
        await adapter.continueConversation(reference, async (context) => {
            await context.sendActivity(req.body.message);
        });
        res.send(200);
    } else {
        res.send(404);
    }
});
```

BotFrameworkで利用される2つのエンドポイント

中身は、**"ボット本体"**を実現する
adapter.**processActivity**(req, res, callback)

中身は、**"外部からの通知"**(プロアクティブメッセージ)を実現する
adapter.**continueConversation**(reference, callback)

/api/messages

中身は、**"ボット本体"を実現する**
adapter.**processActivity**(req, res, callback)

① エンドポイント宛にPOSTコール経由で渡された第1引数(req) のactivityオブジェクトをパースする。

② ①のパースが通ったらcallbackが実行される仕組み。その中でbot.run(context)を実行するようにしておく。

③ ②の変数botには、SDKのActivityHandlerオブジェクトが代入されている。
このオブジェクトのメソッドとして、イベントハンドラ(たとえば、**onMembersAdded**)を定義していく。

/api/notify

中身は、**"外部からの通知"(プロアクティブメッセージ)を実現する**
adapter.**continueConversation**(reference, callback)

① 第1引数のreferenceオブジェクトをパースする。

② ①のパースが通ったらcallbackが実行される仕組み。よって、callback内にメッセージを表示する処理(たとえば、context.sendActivity(msg))を定義していく。

(ポイント) TurnContextオブジェクトを引数とするcallback関数

(ポイント) 同じアダプタ(BotFrameworkAdapter)インスタンスを使用する

/api/notify

【ステップ①】
req.body経由でreference (会話リファレンス) を受け取って
continueConversation の第1引数に渡す

中身は、**"外部からの通知"(プロアクティブメッセージ)を実現する**
BotFrameworkAdapterクラスの
continueConversation(reference, callback) メソッド

「**会話リファレンス**」(reference)
- 「会話」を特定するための識別情報
- 「会話」スレッドのある時点の情報を集めたJSON オブジェクト
- TurnContext.getConversationReference(context.activity) により取得可能

/api/notify エンドポイントに POST HTTPSリクエスト

req.bodyからreferenceを受け取る

referenceを continueConversationの第1引数に渡す

```
server.post('/api/notify', async (req, res) => {
    // Lookup previously saved conversation reference.
    const reference = await findReference(req.body.refId);

    // Proactively notify the user.
    if (reference) {
        await adapter.continueConversation(reference, async (context) => {
            await context.sendActivity(req.body.message);
        });
        res.send(200);
    } else {
        res.send(404);
    }
```

BotFrameworkAdapter class - Azure Bot Service

【ステップ②】
reference (会話リファレンス)のパースが通ったら、
アダプタから受け取ったTurnContextオブジェクト(**context**)を引数に渡して
コールバック関数(**callback**)を実行し、「会話」を再開する。

```
server.post('/api/notify', async (req, res) => {
    // Lookup previously saved conversation reference.
    const reference = await findReference(req.body.refId);

    // Proactively notify the user.
    if (reference) {
        await adapter.continueConversation(reference, async (context) => {
            await context.sendActivity(req.body.message);
        });
        res.send(200);
    } else {
        res.send(404);
    }
```

BotFrameworkAdapter class - Azure Bot Service

TurnContextオブジェクトを引数とするコールバック関数(**callback**)

中身は、"ボット本体"を実現する
adapter.processActivity(req, res, callback)

① エンドポイント宛にPOSTコール経由で渡された第1引数(req) のactivityオブジェクトをパースする。

② ①のパースが通ったらcallback が実行される仕組み。その中でbot.run(context)を実行するようにしておく。

③ ②の変数botには、SDKのActivityHandlerオブジェクトが代入されている。
このオブジェクトのメソッドとして、イベントハンドラ(たとえば、onMembersAdded)を定義していく。

もしも第1引数req (activityを表すJSON)を外部で作成できれば、APIコールにより bot.run をトリガできるが、文脈とタイミングに依存するactivityオブジェクトについて、外部において、SDKで自動的に作成されるものと同じものを作成することは、現実的には困難

第1引数のreference（会話リファレンス）を表すJSONは、
conversationIDとserviceUrlがわかれば作成できる。
あるいは、conversationReferenceをDBに保存しておけば、
そのまま再利用できる。よって、外部からAPIをトリガできる。

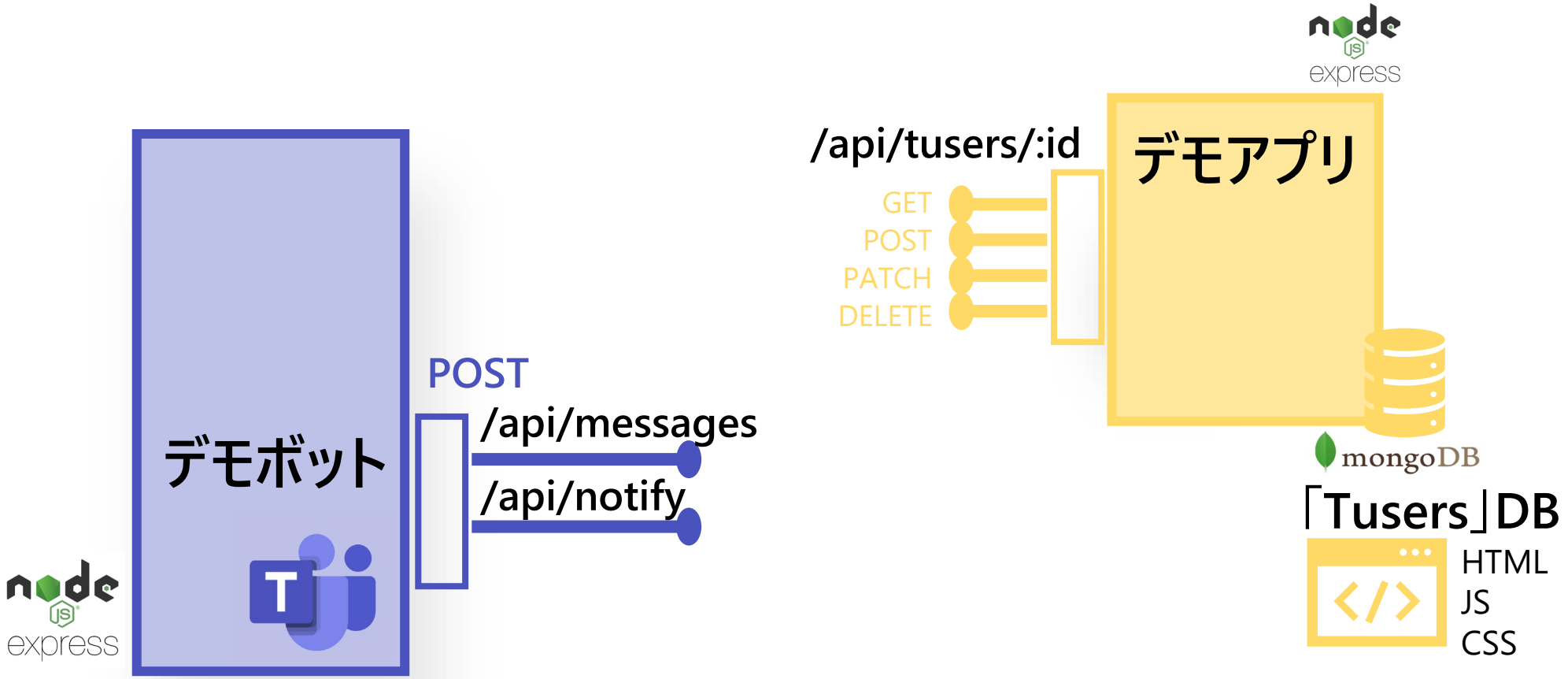
/api/notify

中身は、"外部からの通知"(プロアクティブメッセージ)を実現する
adapter.continueConversation(reference, callback)

① 第1引数のreferenceオブジェクトをパースする。

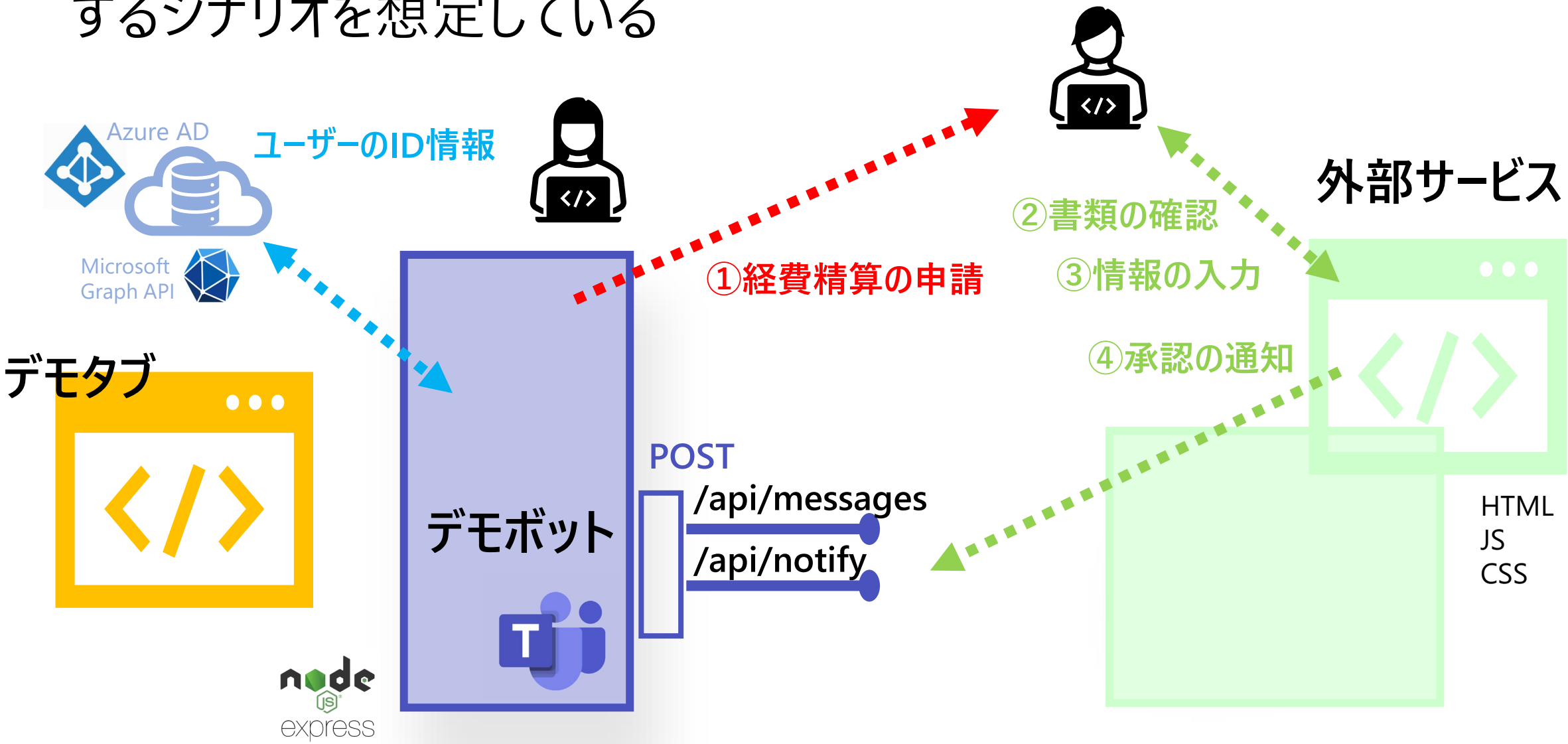
② ①のパースが通ったらcallback が実行される仕組み。よって、callback内にメッセージを表示する処理(たとえば、context.sendActivity(msg))を定義していく。

デモの構成

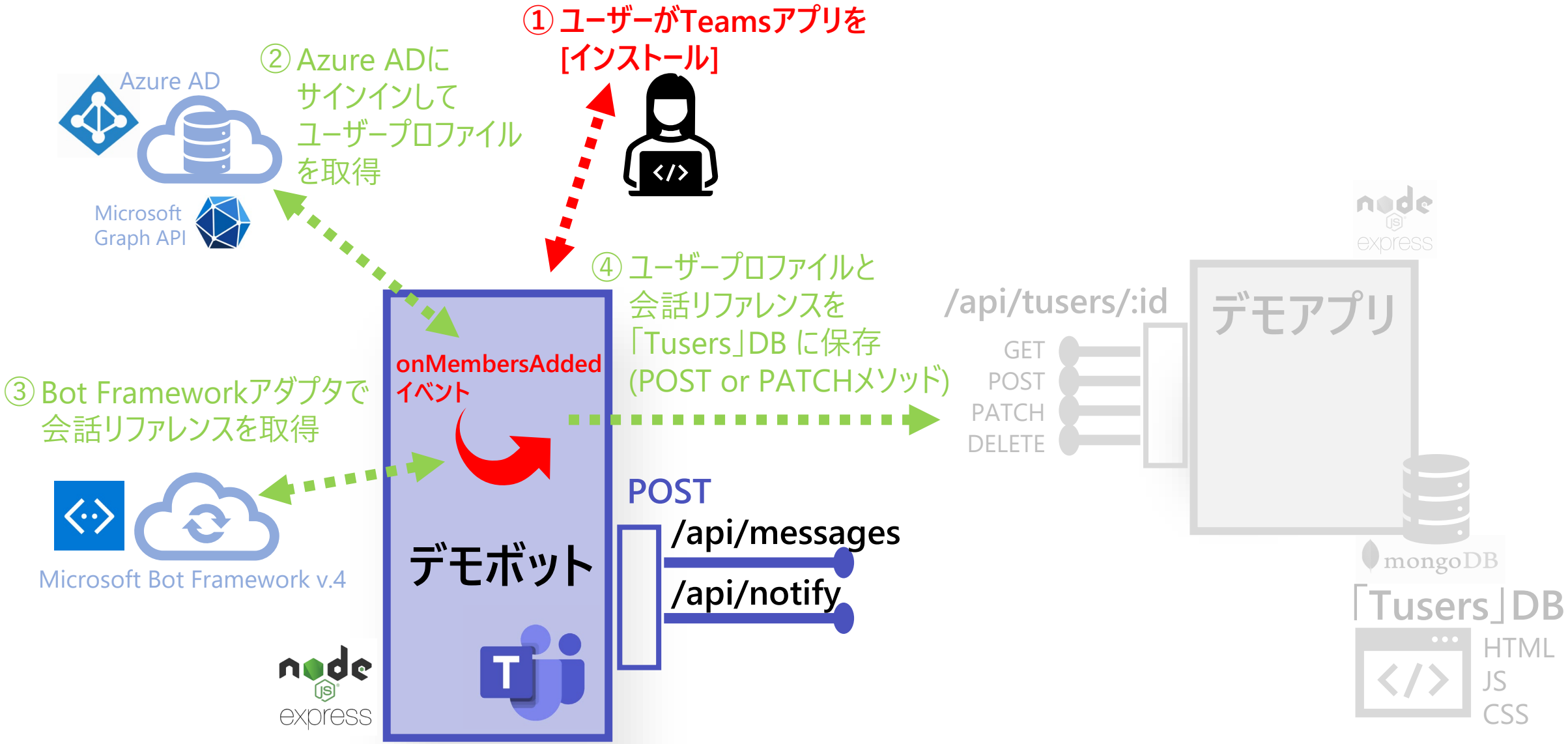


Azure App Registrationsと
Azure Botに登録

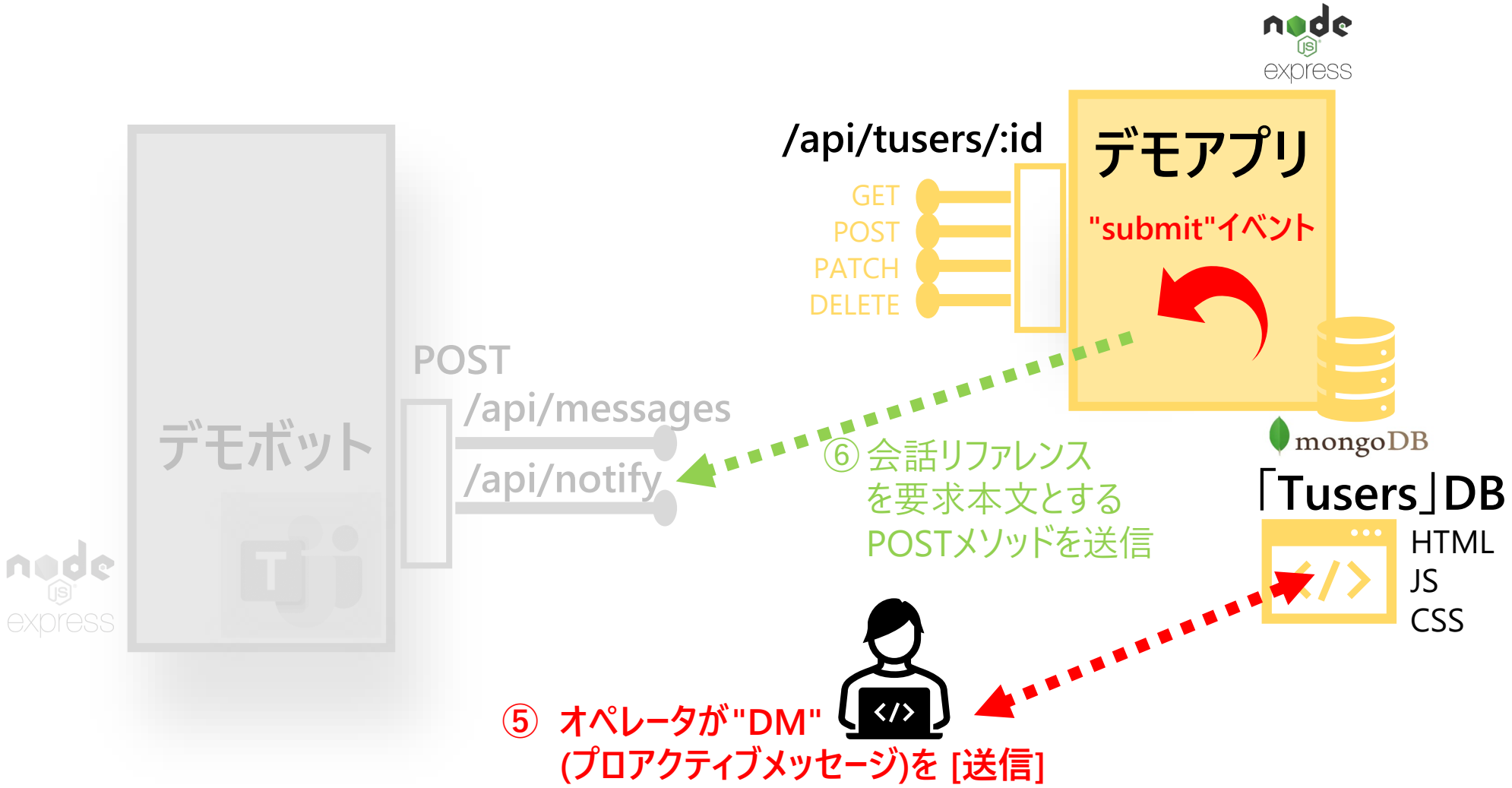
デモアプリは、「外部サービス」と連携するシナリオを想定している



デモ (前半) のステップ


① ユーザーがTeamsアプリを
[インストール]
② Azure ADに
サインインして
ユーザープロファイル
を取得
Azure AD
Microsoft
Graph API
③ Bot Frameworkアダプタで
会話リファレンスを取得
Microsoft Bot Framework v.4
④ ユーザープロファイルと
会話リファレンスを
「Tusers」DB に保存
(POST or PATCHメソッド)
onMembersAdded
イベント
デモボット
node
express
POST
/api/messages
/api/notify
/api/tusers/:id
GET
POST
PATCH
DELETE
デモアプリ
node
express
mongoDB
「Tusers」DB
HTML
JS
CSS

デモ (後半) のステップ


node
express
/api/tusers/:id
デモアプリ
GET
POST
PATCH
DELETE
"submit"イベント
POST
/api/messages
/api/notify
デモボット
⑥ 会話リファレンス
を要求本文とする
POSTメソッドを送信
mongoDB
「Tusers」DB
HTML
JS
CSS
⑤ オペレータが"DM"
(プロアクティブメッセージ)を [送信]

デモのステップまとめ

■ (前半) Teamsにおいて：

① ユーザーがデモボットを[インストール]

onMembersAdded イベントハンドラで

② Azure ADにサインインしてユーザープロファイルを取得

③ Bot Frameworkアダプタから会話リファレンスを取得

④ プロファイルと会話リファレンスを「Tusers」DB に保存
(POST または PATCHメソッド)

■ (後半) デモアプリにおいて:

⑤ オペレータが"DM"(プロアクティブメッセージ)を[送信]

"submit"イベントハンドラで

⑥ 会話リファレンスを要求本文とするPOSTメソッドを/api/notifyに送信

デモのステップ

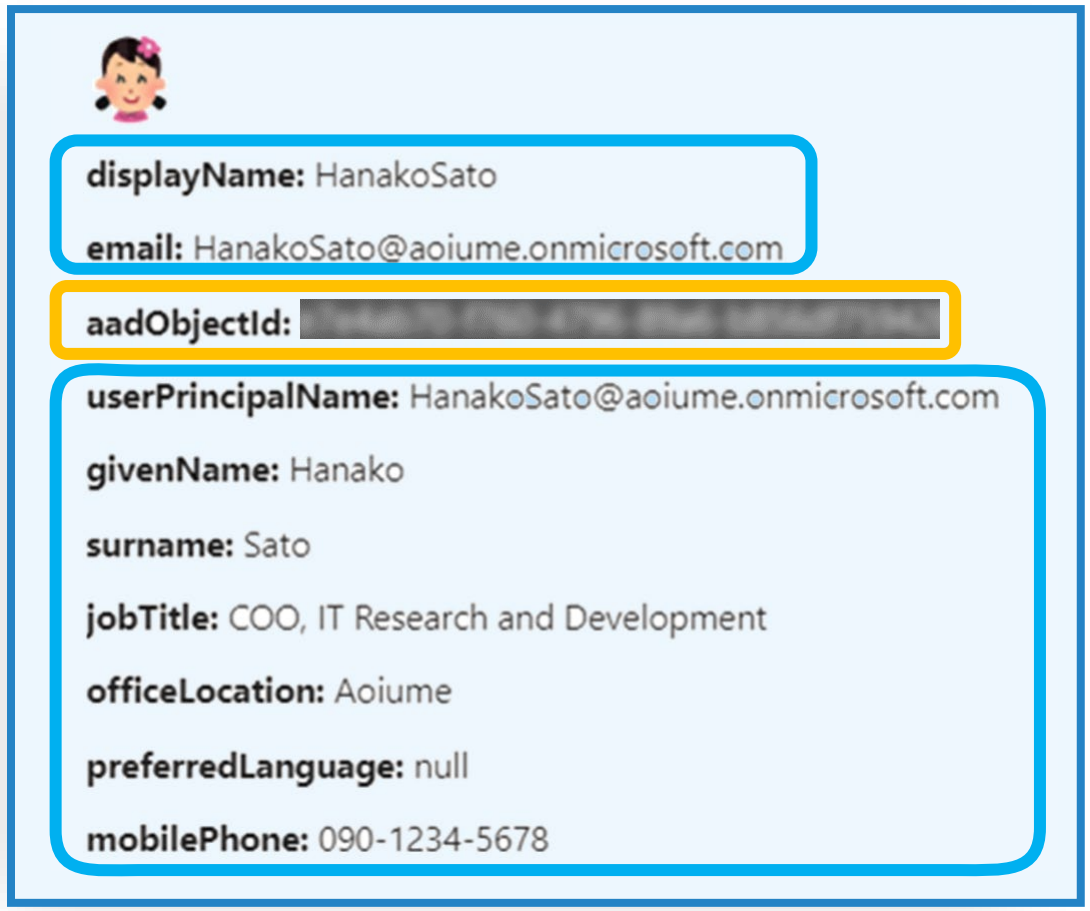
- (前半) Teamsにおいて :
  - ① ユーザーがデモボットを[インストール]

  onMembersAdded イベントハンドラで
  - ② Azure ADにサインインしてユーザープロファイルを取得
  - ③ Bot Frameworkアダプタから会話リファレンスを取得
  - ④ プロファイルと会話リファレンスを「Tusers」DB に保存 (POST または PATCHメソッド)
- (後半) デモアプリにおいて:
  - ⑤ オペレータが"DM"(プロアクティブメッセージ)を[送信]

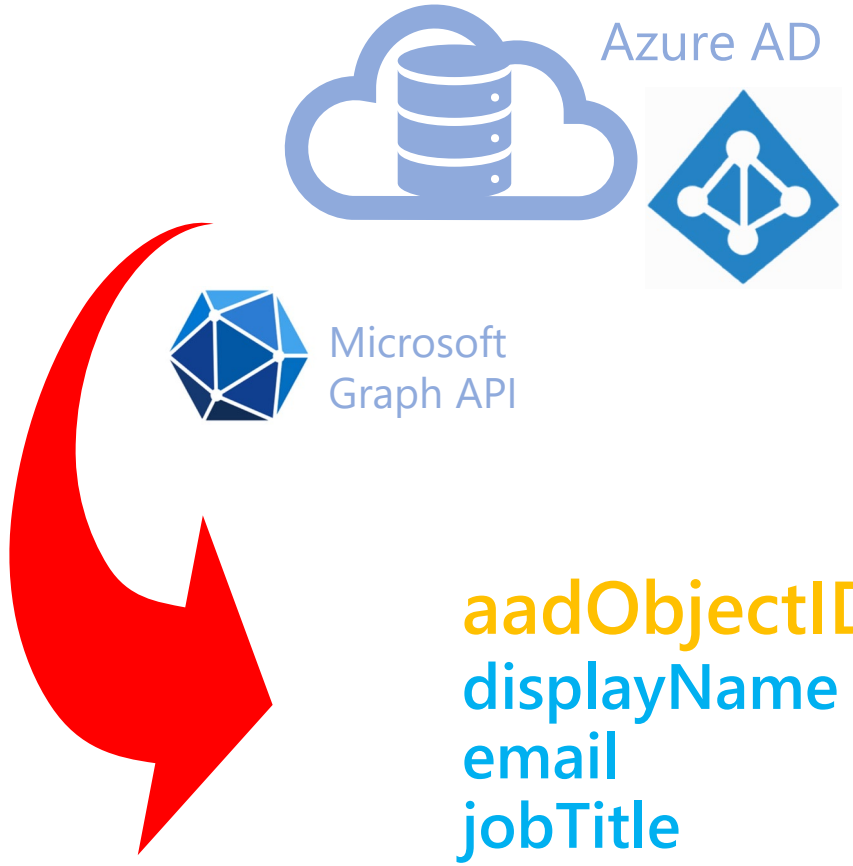
  "submit"イベントハンドラで
  - ⑥ 会話リファレンスを要求本文とするPOSTメソッドを/api/notifyに送信

ステップ②の要点：

Azure ADのユーザープロファイルから、aadObjectID、その他の属性を抽出する



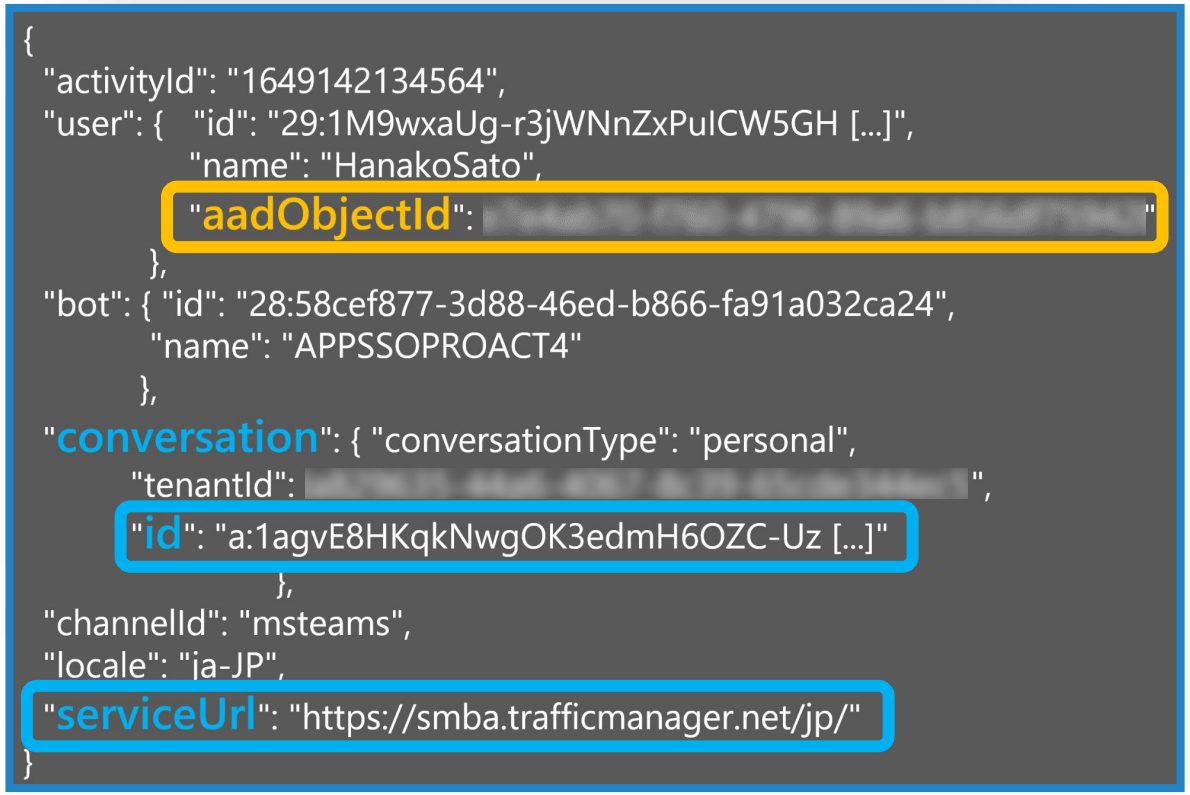
Azure ADのユーザープロファイル



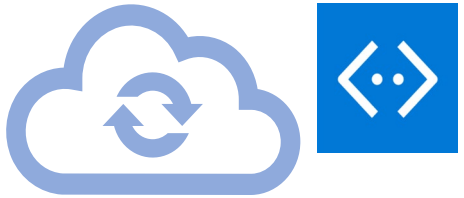
aadObjectID
displayName
email
jobTitle
officeLocation ...

ステップ③の要点:

会話リファレンスから aadObjectID、conversationID、serviceUrlを抽出する



会話リファレンス (conversationReference)



Microsoft Bot Framework v.4

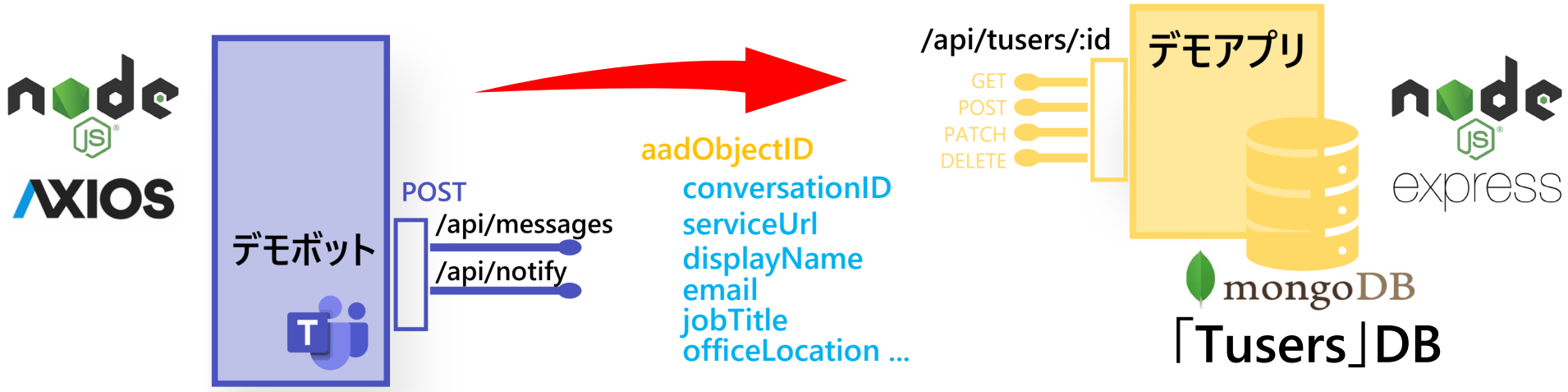
Bot Framework SDK
アダプタ

aadObjectID
conversationID
serviceUrl

ステップ④の要点：

抽出した属性値を、aadObjectIDを主キーとして「Tusers」DBに保存する

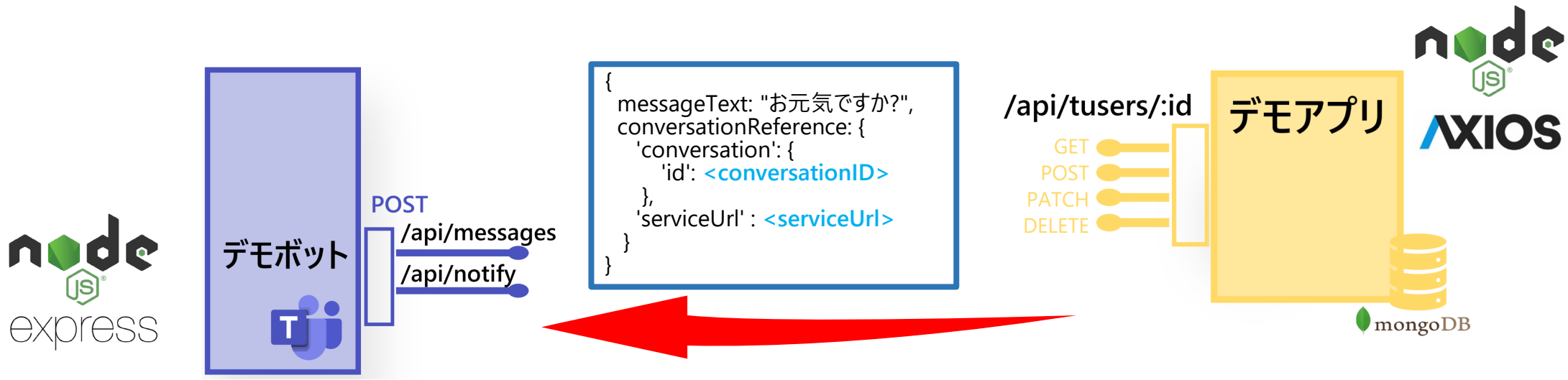
1. ボット側からは、属性値を要求本文に入れ、aadObjectIDを:idとして、/api/tusers/:id宛てにPOST (追加) または PATCH (更新) HTTPリクエストを送信する。
2. アプリ側では、/api/tusers/:id宛てのGET/POST/PATCH/DELETE HTTPリクエストに対して、「Tusers」DBのレコードを検索/追加/更新/削除するハンドラを実装しておく (RESTfulなWebAPI) 。



ステップ⑥の要点:

ユーザーに向けてプロアクティブメッセージ("DM")を送信する

1. アプリ側からは、「会話リファレンス」(会話リファレンス全体、またはconversationIDとserviceUrlだけを含むもの)、およびメッセージテキストを要求本文に入れ、/api/notify 宛てにPOST HTTP リクエストを送信する。
2. ボット側では、/api/notify宛てのPOST HTTPリクエストに対して、「会話リファレンス」を引数にしてcontinueConversation()を実行するハンドラを実装しておく。



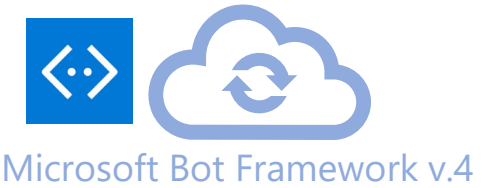
今回のデモを通してご説明したこと

- 「会話リファレンス」を保存して「会話の継続」により"DM"(プロアクティブメッセージ)を送信する方法
- 

と 

の類似性と違い
- TurnContextオブジェクトを使用した、Activityイベントハンドラの実装方法 (例: onMembersAdded メソッド)

Bot Framework Ver.4



「SDKによる実装」を行うように設計されている

「SDKによる実装」 > 「APIコールによる実装」

素直　簡単　高速

※ 外部サービスからユーザーに通知を送るために、処理のトリガにのみAPIコールを使用する (例: 経費精算の承認依頼など)。