

# 受付スキルについて

マネージメントサービス株式会社 風間玲央

# 受付スキルの概要

社員番号(数字)を聞き取って、Googleスプレッドシートの指定したシートに出欠ステータスを  
入力するスキルです。社内イベントの受付を想定しています。

もちろん、入出力の「社員番号」という言葉を受付番号・ID等に変更すれば、その他のイベント  
にも使えます。

AWS Lambda関数とGoogle Apps Scriptのソースコード、インテントのCSVファイル、その他設定  
のスクリーンショット等は以下のリポジトリにあります。

<https://github.com/forshoes-admin/Alexa-reception>

また、ハンズオン資料もあります。

こちらでは、対話モデルをより簡略化し、alexa developer consoleの詳しい使い方も載せてい  
ます。

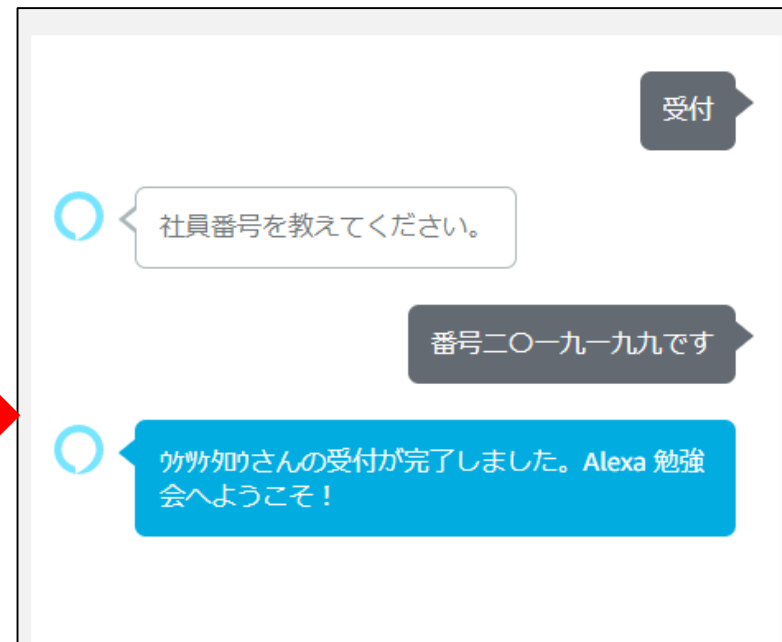
<https://github.com/forshoes-admin/Alexa-reception/tree/master/hands-on>

# スキルの使い方

## スキルの動作

[社員番号(は)(が)／番号(は)]+[社員番号8桁または7桁]+[です／番(で)(です)]と発話して社員番号を受け取り、Googleスプレッドシートの出欠ステータスにチェック(値をTRUEに)します。

社員番号は8桁ですが、先頭の0を省略した7桁でも受け取ることができます。Googleスプレッドシート上では先頭の0を省略しているため、先頭の0はあってもなくても受付できます。



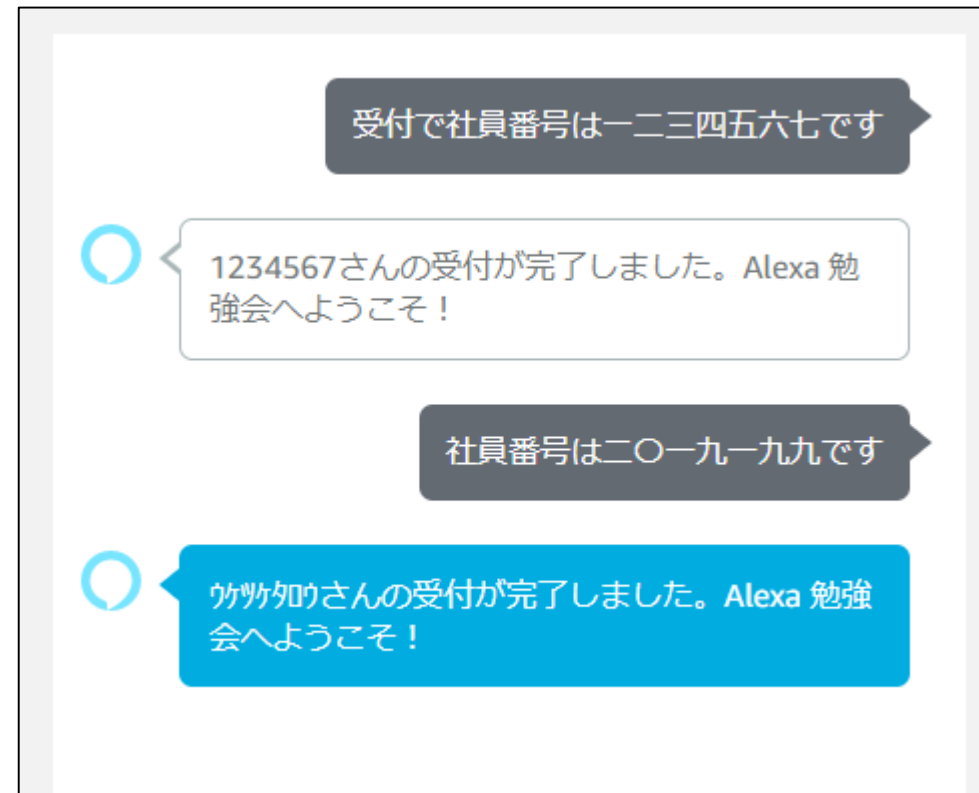
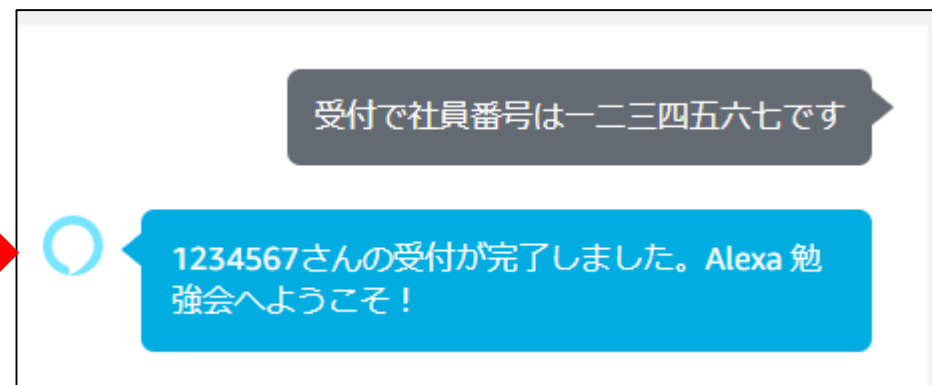
出欠シート (Alexa連携)				
ファイル 編集 表示 挿入 表示形式 データ ツール アド				
fx				
	A	B	C	D
1	出欠ステータス	社員番号	名前	フリガナ
2	<input checked="" type="checkbox"/>	2019199	ウケツケタロウ	ウケツケタロウ
3	<input type="checkbox"/>	2030001	ウケツケハナコ	ウケツケハナコ
4	<input checked="" type="checkbox"/>	2019198	ウケツケジロウ	ウケツケジロウ
5	<input type="checkbox"/>	11000011	ウケツケサブロウ	ウケツケサブロウ
6	<input type="checkbox"/>	11010022	ウケツケシロウ	ウケツケシロウ
7	<input type="checkbox"/>	1234567	01234567	01234567
8				

## スキルの呼び出し方

呼び出し名单体で呼び出すだけでなく、「受付で社員番号は1234567です」のように、[呼び出し名]+[で]+[アクション]でスキルを呼び出すと同時に受付をすることもできます。

参考: ユーザーによるカスタムスキルの呼び出し  
(<https://developer.amazon.com/ja/docs/custom-skills/understanding-how-users-invoke-custom-skills.html>)

また、受付を開始した後は、「受付で」の呼び出し名を言うことなく、連続して受け付けることができます。

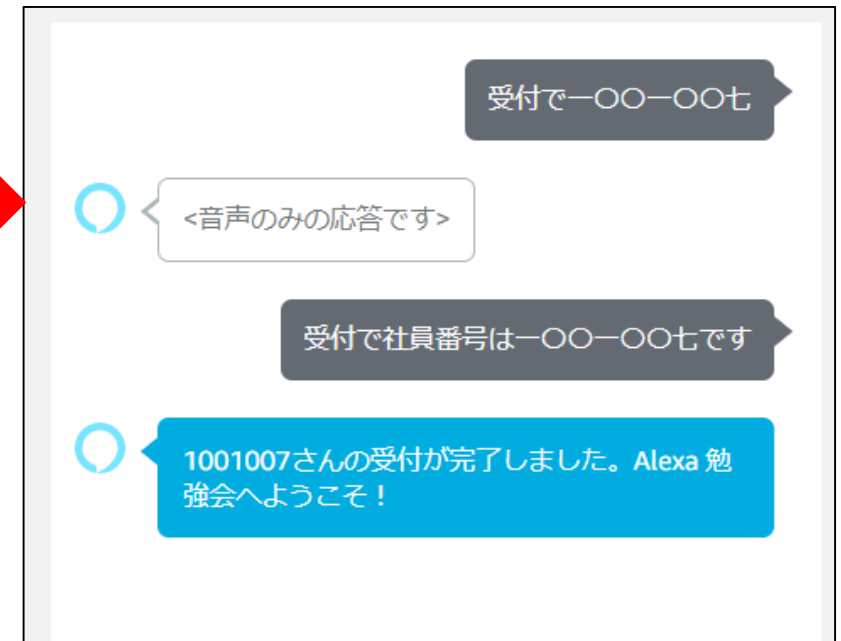
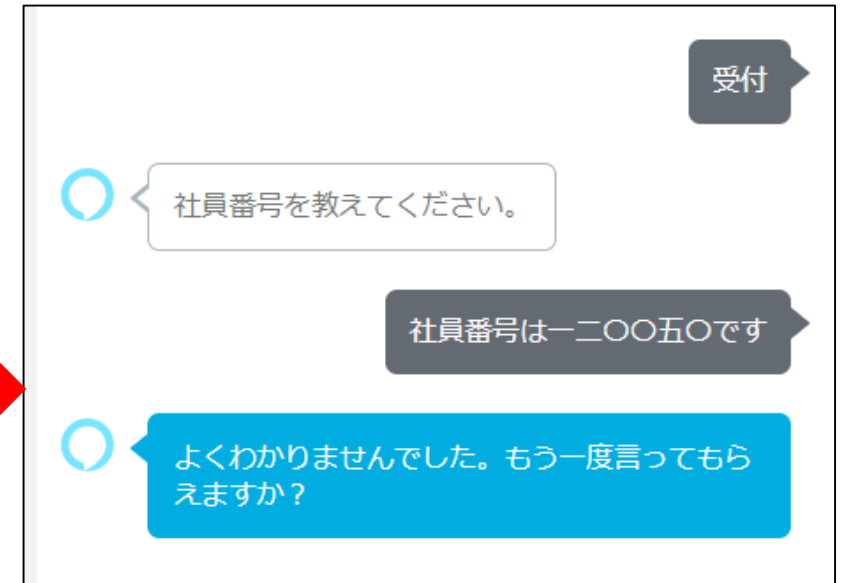


## スキルの注意点

6桁以下の数字はうまく受け取れません。受け取りたい場合は、6桁以下のサンプル発話を追加します。

数字のみのサンプル発話も登録していますが、0が3個以上連続していたり、0の連続が2回以上あると数字のみでは数字を受け取れません。

前後に「社員番号は」と「です」をつけると受け取れます。



## Googleスプレッドシートの設定①

A列にチェックボックス([※1:参考記事](#))、B列に社員番号、D列にフリガナを入力します。

シートでは、「`=A1=TRUE`」の条件で条件付き書式([※2:参考記事](#))を設定し、出欠ステータスがTRUEのときに色がつくようにしました。



	A	B	C	D
1	出欠ステータス	社員番号	名前	フリガナ
2	<input checked="" type="checkbox"/>	2019199	ウケツケタロウ	ウケツケタロウ
3	<input type="checkbox"/>	2030001	ウケツケハナコ	ウケツケハナコ
4	<input checked="" type="checkbox"/>	2019198	ウケツケシロウ	ウケツケシロウ
5	<input type="checkbox"/>	11000011	ウケツケサブロウ	ウケツケサブロウ
6	<input type="checkbox"/>	11010022	ウケツケシロウ	ウケツケシロウ
7	<input type="checkbox"/>	1234567	01234567	01234567

条件付き書式設定ルール

123 「`=A1=TRUE`」である  
カスタムの数式  
A1:D1001

+ 条件を追加

また、表示形式をカスタム数値形式([※3:参考記事](#))の「0」にしています。これにより、先頭の0は自動で消去され、整数で表示されます。



カスタム数値形式

0 適用

サンプル: 1235 ヘルプ

0	1235
###0	1235
0.00	1234.56

## Googleスプレッドシートの設定②

さらに、F1列に値を入力すると、「**Alexa勉強会**へようこそ！」のように、Alexaがイベント名を読み上げます。



E	F
Alexaが読み上げるイベント名または団体名を入力（省略可）⇒	Alexa勉強会

設定が終わったら、使うシートを1番目にしておきます。



+	Alexa勉強会 ▼	全社員 ▼	2019/8/16 ▼	出欠シート ▼	読み仮名 ▼	出欠シ	◀ ▶
---	------------	-------	-------------	---------	--------	-----	-----



# Google スプレッドシートについて参考になる記事

※1: チェックボックスを追加して使用する

(<https://support.google.com/docs/answer/7684717?co=GENIE.Platform%3DDesktop&hl=ja>)

※2: Google スプレッドシートで条件付き書式ルールを使用する

(<https://support.google.com/docs/answer/78413?co=GENIE.Platform%3DDesktop&hl=ja>)

※3: スプレッドシートで数値の表示形式を設定する

(<https://support.google.com/docs/answer/56470?co=GENIE.Platform%3DDesktop&hl=ja>)

# スキルの作成方法①

## Google Apps Script

## ソースコードについて①

以下の記事に載っているfindParticipant関数と、doGet関数を修正したものです。

【Alexa】Connpass用の受付スキルを作ってみた #Alexa #AlexaDevs  
(<https://dev.classmethod.jp/voice-assistant/try-to-develop-registration-skill/>)

findParticipant関数について、修正した部分は以下の通りです。

F1セルの値を取得して、空白でなければ「へ」をつけて渡します。

```
var iventName = sheet.getRange("F1").getValue();
    if (iventName) {
        iventName += "へ";
    }
```

「[4列目の値]+さんの受付が完了しました。+[イベント名へ]+ようこそ！」をAlexaに渡します。4列目の値は、漢字だと正確に読めない場合があるため、フリガナを設定しておきます。F1セルが空白の場合、二つ目の文は単に「ようこそ！」になります。

```
return values[i][3] + "さんの受付が完了しました。" + iventName + "ようこそ！";
```

## ソースコードについて②

背景色の変更は条件付き書式で設定したため、以下のコードを削除しました。

```
rng.setBackground("palegreen");
```

これは、手動でチェックしたときにも背景色を変えられるようにするためです。

doGet関数については、シート取得のコードを以下のようにしました。

```
var sheet = SpreadsheetApp.getActiveSpreadsheet().getSheets()[0];
```

対象のシートの名前で取得するのではなく、1番目のシートを取得するようにしました。

対象のシートを1番目に設定しておけば、イベント後もシート名を変えることなく、次のイベントで使えます。

## Google スプレッドシートへの追加

ソースコードは以下の場所にあります。

<https://github.com/forshoes-admin/Alexa-reception/tree/master/GAS>

このうち、findParticipant.gsがAlexaから呼び出すソースコードです。まず、Googleスプレッドシートの画面から、[ツール] > [スクリプトエディタ]でプロジェクトを作成します。

コード.gsにfindParticipant.gsをコピーして上書きで貼り付けるか、[ファイル] > [新規作成] > [スクリプトファイル]で新しいスクリプトファイルを作成して上書きで貼り付けます。

フリガナをシートに追加するaddKana.gsを使う場合は、ソースコードをコピーして新しいスクリプトファイルに上書きで貼り付けます。

The first screenshot shows the Google Sheets interface for a spreadsheet titled '出欠シート (Alexa連携)'. The 'Tools' menu is open, and 'スクリプトエディタ' (Script Editor) is highlighted. A red arrow points from the text 'プロジェクトを作成します' to this menu item.

The second screenshot shows the 'Script Editor' interface. The project is named '無題のプロジェクト' (Untitled Project). The code editor shows a template function: 

```
function myFunction() {  
  |  
}
```

 A red box highlights this code block.

The third screenshot shows the 'File' menu in the 'Script Editor' interface. The '新規作成' (New) option is selected, and the 'スクリプト ファイル' (Script File) option is highlighted in the submenu. A red arrow points from the text '新しいスクリプトファイルを作成して' to this option.

# プロジェクトを公開

[公開] > [ウェブアプリケーションとして導入]を選択します。

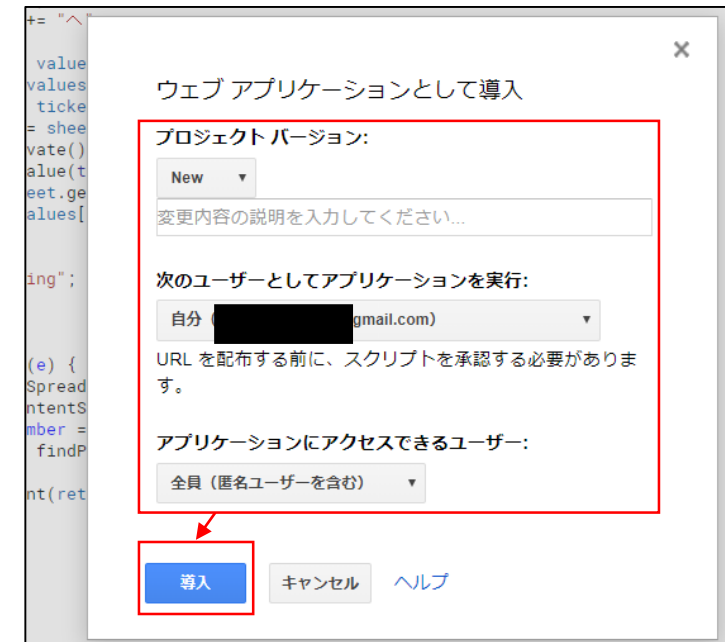
プロジェクト バージョン:New

次のユーザーとしてアプリケーションを実行:自分  
(xxx@gmail.com)

アプリケーションにアクセスできるユーザー:全員  
(匿名ユーザーを含む)

に設定して、「導入」ボタンをクリックします。

アプリケーションには誰でもアクセスできるので、URLの管理には気をつけてください。また、電話番号・メールアドレス等の重要な情報は扱わないようにしましょう。



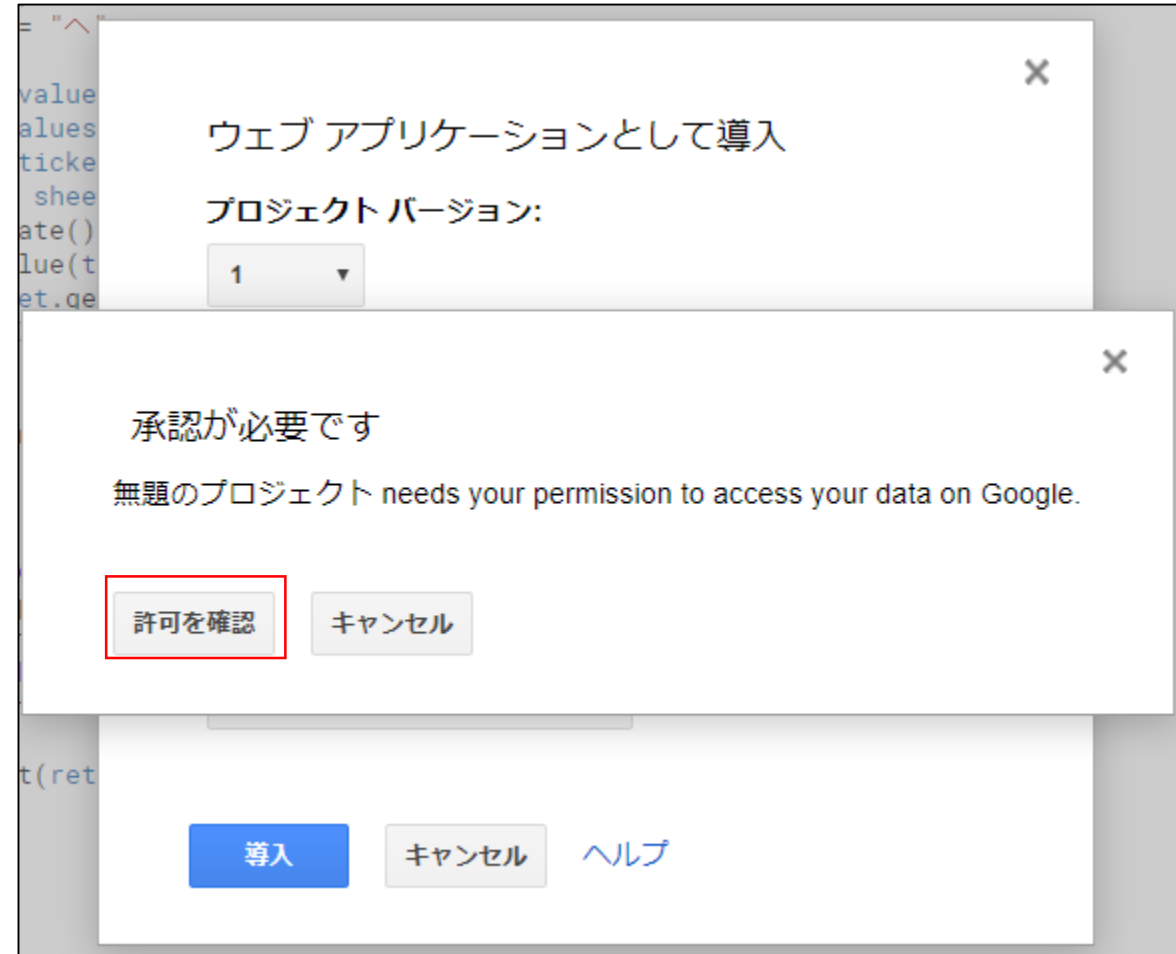
# ログイン

アカウントをクリックしてログインします。



## 許可を確認

「許可を確認」ボタンをクリックします。





詳細へ

「詳細」リンクをクリックします。



このアプリは確認されていません

このアプリは、Google による確認が済んでいません。よく知っている信頼できるデベロッパーの場合に限り続行してください。

詳細

安全なページに戻る

プロジェクト(安全ではないページ)に移動

「[プロジェクト名](安全ではないページ)に移動」リンクをクリックします。



## このアプリは確認されていません

このアプリは、Google による確認が済んでいません。よく知っている信頼できるデベロッパーの場合に限り続行してください。

[詳細を非表示](#)[安全なページに戻る](#)

Google ではまだこのアプリを確認していないため、アプリの信頼性を保証できません。未確認のアプリは、あなたの個人データを脅かす可能性があります。

[詳細](#)

無題のプロジェクト (安全ではないページ) に移動

# 許可

「許可」ボタンをクリックします。



## ウェブ アプリケーションとして導入完了

「現在のウェブ アプリケーションの URL」を控えて、「OK」ボタンをクリックします。

ポップアップを閉じた後、URL は[公開] > [ウェブ アプリケーションとして導入]で確認できます。



## スキルの作成方法②

### 対話モデル

# 作成する対話モデルについて

\_intentの下にスロットがあり、スロットで数字を受け取るという構造は、以下の記事を参考にしています。

【Alexa】Connpass用の受付スキルを作ってみた #Alexa #AlexaDevs  
(<https://dev.classmethod.jp/voice-assistant/try-to-develop-registration-skill/>)

しかし、上記記事で使われている「AMAZON.FOUR\_DIGIT\_NUMBER」というスロットタイプでは、受付番号に0が連続すると受け取れないことがありました。

そこで以下の記事を参考に、8桁または7桁の数字を1桁ずつ受け取るようにしました。

[日本語Alexa] 4桁の数字を確実に受けとるためには ～より自然に会話できるスキル作成のために～

(<https://dev.classmethod.jp/cloud/4-digit-number-custom-slot/>)

## スロットタイプを追加

「スロットタイプ」の「追加」または「+スロットタイプ」をクリックします。

「LIST\_OF\_NUMBER」と入力して「カスタムスロットタイプを作成」をクリックします。

The screenshot shows the Amazon Lexa console interface. On the left sidebar, under the 'Custom' (カスタム) section, the 'Slot Types' (スロットタイプ) option is selected, showing 0 slot types. A red box highlights the '+ Add' (追加) button next to it. Another red box highlights the '+ Slot Type' (+スロットタイプ) button in the main area. Below this, the 'Add Slot Type' (スロットタイプを追加) screen is shown. The 'Custom Slot Type' (カスタムスロットタイプを作成) radio button is selected. The text input field contains 'LIST\_OF\_NUMBER'. A red box highlights the 'Create Custom Slot Type' (カスタムスロットタイプを作成) button. The 'Use Built-in Slot Type' (Alexaのビルトインライブラリから既存のスロットタイプを使用) option is also visible but not selected.

## スロット値を追加

以下の場所にあるスロットタイプのCSVファイルを「一括編集」から取り込むことができます。

<https://github.com/forshoes-admin/Alexa-reception/tree/master/slot>

これは、0～9までの数字を取得するためのものです。なお、Alexaでは、日本語の数字は漢数字で扱います。

The screenshot shows the Alexa Developer Console interface. On the left sidebar, under 'カスタム' (Custom), the 'スロットタイプ(1)' (Slot Type (1)) section is expanded, showing 'LIST\_OF\_NUMBER' selected. The main area displays the configuration for 'LIST\_OF\_NUMBER'. At the top, it says 'スロットのタイプ / LIST\_OF\_NUMBER'. Below this, there's a section for 'スロット値(0)' (Slot Value (0)) with a red box highlighting the '一括編集' (Bulk Edit) button. A search bar is also present. The main content area shows a message: 'このスロットタイプにはスロット値がありません' (There are no slot values for this slot type). Below this, a note explains that for custom slot types, a list of values must be defined. At the bottom, there's a table header for '(LIST\_OF\_NUMBER)を使用中のスロット' (Slots using (LIST\_OF\_NUMBER)) with columns for 'スロット名' (Slot Name) and 'インテント' (Intent), but the table is empty with a message 'インテントスロットが見つかりません' (No intent slots found).

カスタム

対話モデル

呼び出し名

✓ インテント(5) + 追加

- HelloWorldIntent
- ✓ ビルトインインテント(4)
- AMAZON.CancelIntent
- AMAZON.HelpIntent
- AMAZON.StopIntent
- AMAZON.NavigateHomeIntent

✓ スロットタイプ(1) + 追加

LIST\_OF\_NUMBER

JSONエディター

インターフェース

エンドポイント

インテント履歴

画面表示 ベータ

スキル内商品

アカウントリンク

アクセス権限

Japanese (日本語)

スロットのタイプ / LIST\_OF\_NUMBER

スロット値(0) 一括編集 書き出し 検索

このスロットタイプの新しい値を入力 +

このスロットタイプにはスロット値がありません

カスタムスロットタイプでは、このスロットで使用する値のリストを定義します。これはAmazonのビルトインタイプセットでサポートされていない項目に使用されるもので、ほとんどのケースに推奨されます。

(LIST\_OF\_NUMBER)を使用中のスロット

スロット名	インテント
インテントスロットが見つかりません	

© 2010 - 2019, Amazon.com, Inc. or its affiliates. All Rights Reserved. 無断複写・転載を禁じます。 利用規約 ドキュメント フォーラム Alexa開発者ブログ Alexa開発者ホーム



## スロットタイプを保存

スロット値を追加したら、「モデルを保存」をクリックします（保存しないと、セッションの有効期限が切れたときにやり直しになります）。

The screenshot shows the Amazon Lex console interface for configuring a slot type. The left sidebar contains navigation options: カスタム (Custom), 対話モデル (Dialog Model), 呼び出し名 (Invocation Name), インテント(5) (Intent (5)), ビルトインインテント(4) (Built-in Intent (4)), スロットタイプ(1) (Slot Type (1)), JSONエディター (JSON Editor), インターフェース (Interface), エンドポイント (Endpoint), インテント履歴 (Intent History), 画面表示 (Screen Display), スキル内商品 (Skill In-Context), and アカウントリンク (Account Link). The 'スロットタイプ(1)' section is expanded, showing 'LIST\_OF\_NUMBER' selected. The main area displays the 'スロットのタイプ / LIST\_OF\_NUMBER' configuration page. At the top, there are buttons for 'モデルを保存' (Save Model), 'バージョン' (Version), 'モデルをビルド' (Build Model), and '発話プロファイラー' (Utterance Profiler). Below these, the 'スロット値(10)' (Slot Values (10)) section is visible, with a search bar and a table of slot values. The table has columns for '値' (Value), 'ID (オプション)' (ID (Optional)), and '同義語(オプション)' (Synonyms (Optional)). The values range from 九 (9) to 二 (2). Each row has a '+ 同義語を追加' (Add Synonym) button and a trash icon.

日本語

モデルを保存 バージョン モデルをビルド 発話プロファイラー

カスタム

対話モデル

呼び出し名

インテント(5) + 追加

HelloWorldIntent

ビルトインインテント(4)

AMAZON.CancelIntent

AMAZON.HelpIntent

AMAZON.StopIntent

AMAZON.NavigateHomeIntent

スロットタイプ(1) + 追加

LIST\_OF\_NUMBER

JSONエディター

インターフェース

エンドポイント

インテント履歴

画面表示 ベータ

スキル内商品

アカウントリンク

スロットのタイプ / LIST\_OF\_NUMBER

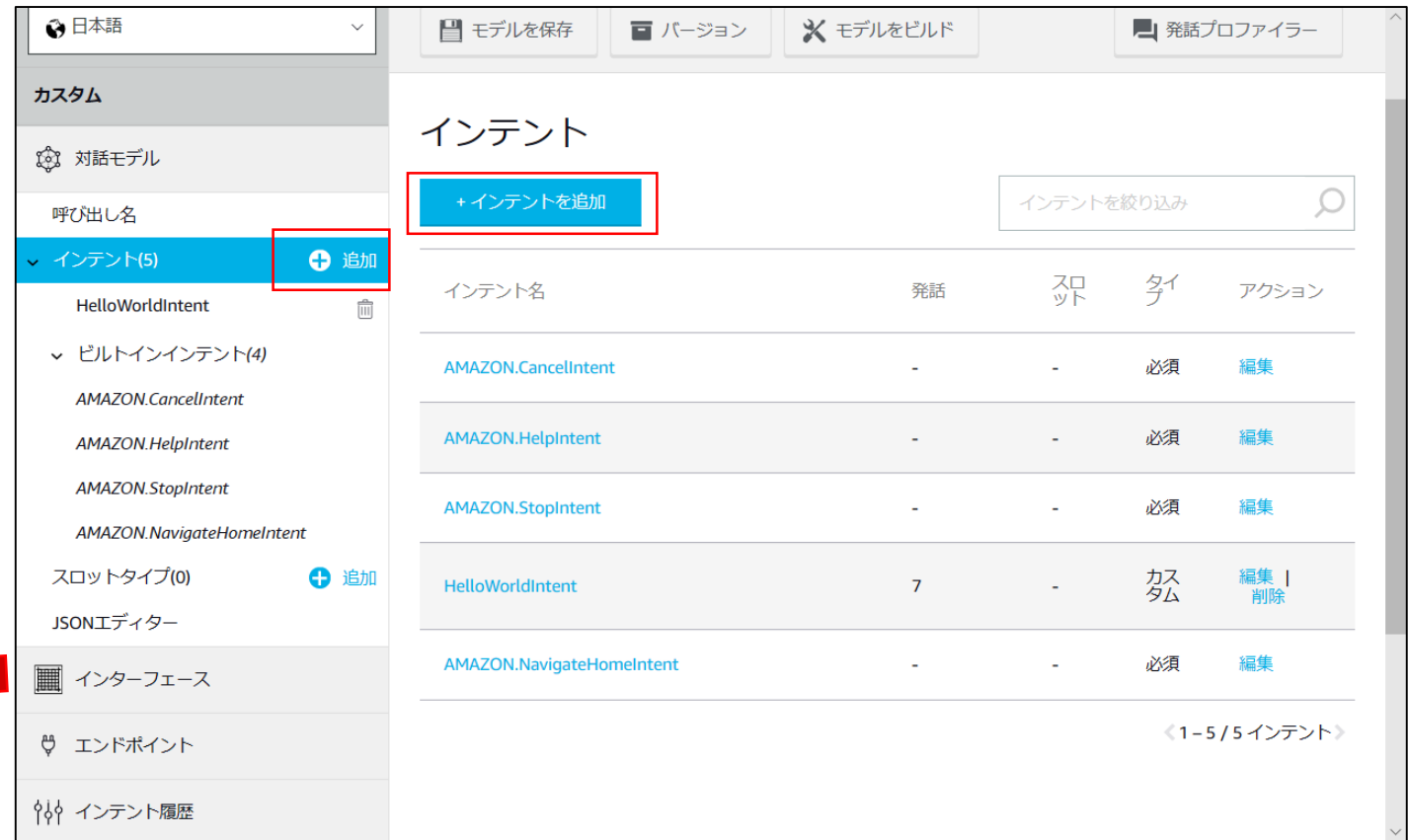
スロット値(10) 一括編集 書き出し 検索

このスロットタイプの新しい値を入力 +

値	ID (オプション)	同義語(オプション)	
九	9	同義語を追加	+ 削除
八	8	同義語を追加	+ 削除
七	7	同義語を追加	+ 削除
六	6	同義語を追加	+ 削除
五	5	同義語を追加	+ 削除
四	4	同義語を追加	+ 削除
三	3	同義語を追加	+ 削除
二	2	同義語を追加	+ 削除

## \_intentを追加

次に、「intent」の「追加」または「+intentを追加」から、「employeeNumber」と入力して「カスタムintentを作成」をクリックします。



日本語

モデルを保存 バージョン モデルをビルド 発話プロファイラー

### intent

+intentを追加

intentを絞り込み

intent名	発話	スロット	タイプ	アクション
AMAZON.CancelIntent	-	-	必須	編集
AMAZON.HelpIntent	-	-	必須	編集
AMAZON.StopIntent	-	-	必須	編集
AMAZON.NavigateHomeIntent	-	-	必須	編集
HelloWorldIntent	7	-	カスタム	編集   削除
AMAZON.NavigateHomeIntent	-	-	必須	編集

< 1 - 5 / 5 intent >



### intentを追加

intentとは、ユーザーが話しかけたリクエストを実行するアクションのことです。intentについての詳細は[こちら](#)。

☒ カスタムintentを作成 ②

employeeNumber カスタムintentを作成

☐ Alexaのビルトインライブラリから既存のintentを使用 ③

ビルトインintentの使用についての詳細は[こちら](#)。

## サンプル発話を追加

以下の場所にあるIntentのCSVファイルを「一括編集」から取り込むことができます。

<https://github.com/forshoes-admin/Alexa-reception/tree/master/intent>

カスタム

対話モデル

呼び出し名

Intent (6) + 追加

- HelloWorldIntent
- employeeNumber
- ビルトインIntent (4)
- AMAZON.CancelIntent
- AMAZON.HelpIntent
- AMAZON.StopIntent
- AMAZON.NavigateHomeIntent

スロットタイプ (0) + 追加

JSONエディター

インターフェース

エンドポイント

Intent履歴

画面表示 ベータ

スキル内商品

アカウントリンク

アクセス権限

### Intent / employeeNumber

サンプル発話 (0) 一括編集 書き出し

このIntentの呼び出しに使われると考えられる発話 +

このIntentにはサンプル発話がありません

サンプル発話は、Intentを呼び出すためにユーザーが話しかけるフレーズのことです。

< 0 - 0 / 0 >

#### ダイアログデリゲートのルール

このIntentでは、ダイアログ管... > これが無効にされている理由

#### Intentスロット (0)

順序	名前	スロットタイプ	アクション
1	新しいスロットを作成	+ スロットタイプを選択	ダイアログを編集   削除

## 使用するサンプル発話について

このサンプル発話は、「社員番号は[数字8桁]です」の発話を基本に、60個のパターンに派生させたものです。{first\_number}から{eighth\_number}または{seventh\_number}まで、1桁ずつスロットで数字を取得します。

サンプル発話は必要に応じて画面から、またはファイルをテキストエディタで編集して使ってください。

	{sixth_number} {seventh_number} です	
社員番号が	{first_number} {second_number} {third_number} {fourth_number} {fifth_number}	
	{sixth_number} {seventh_number} 番	
社員番号が	{first_number} {second_number} {third_number} {fourth_number} {fifth_number}	
	{sixth_number} {seventh_number} 番で	
社員番号が	{first_number} {second_number} {third_number} {fourth_number} {fifth_number}	
	{sixth_number} {seventh_number} 番です	
社員番号が	{first_number} {second_number} {third_number} {fourth_number} {fifth_number}	
	{sixth_number} {seventh_number}	
社員番号	{first_number} {second_number} {third_number} {fourth_number} {fifth_number}	
	{sixth_number} {seventh_number} です	
社員番号	{first_number} {second_number} {third_number} {fourth_number} {fifth_number}	
	{sixth_number} {seventh_number} 番	
社員番号	{first_number} {second_number} {third_number} {fourth_number} {fifth_number}	
	{sixth_number} {seventh_number} 番で	
社員番号	{first_number} {second_number} {third_number} {fourth_number} {fifth_number}	
	{sixth_number} {seventh_number} 番です	
社員番号	{first_number} {second_number} {third_number} {fourth_number} {fifth_number}	
	{sixth_number} {seventh_number}	
番号	{first_number} {second_number} {third_number} {fourth_number} {fifth_number} {sixth_number}	
	{seventh_number} です	
番号	{first_number} {second_number} {third_number} {fourth_number} {fifth_number} {sixth_number}	
	{seventh_number} 番	
番号	{first_number} {second_number} {third_number} {fourth_number} {fifth_number} {sixth_number}	

## スロットタイプを選択

「first\_number」から  
「eighth\_number」まで、すべての  
\_intentスロットのスロットタイ  
プに「LIST\_OF\_NUMBER」を選択  
して「モデルを保存」しておきま  
す。

The screenshot shows the Amazon Lex console interface for configuring an intent. The left sidebar contains navigation options: 'エンドポイント' (Endpoints), 'intent履歴' (Intent History), '画面表示' (Screen Display), 'ベータ' (Beta), 'スキル内商品' (Items within skill), 'アカウントリンク' (Account Link), and 'アクセス権限' (Access Permissions). The main area displays a list of intent slots, numbered 2 through 9. Each slot has a color-coded label (e.g., 'second\_number', 'third\_number', etc.) and a dropdown menu set to 'LIST\_OF\_NUMBER'. A red box highlights the dropdown for slot 8, which is currently open, showing 'LIST\_OF\_NUMBER' as the selected option. Below slot 8, there is a button labeled '新しいスロットを作成' (Create new slot) with a plus sign. At the bottom, there is a section labeled 'intentの確認' (Confirm intent).

Slot Number	Slot Label	Slot Type	Action
2	second_number	LIST_OF_NUMBER	ダイアログを編集   削除
3	third_number	LIST_OF_NUMBER	ダイアログを編集   削除
4	fourth_number	LIST_OF_NUMBER	ダイアログを編集   削除
5	fifth_number	LIST_OF_NUMBER	ダイアログを編集   削除
6	sixth_number	LIST_OF_NUMBER	ダイアログを編集   削除
7	seventh_number	LIST_OF_NUMBER	ダイアログを編集   削除
8	eighth_number	LIST_OF_NUMBER	ダイアログを編集   削除
9	新しいスロットを作成	+	ダイアログを編集   削除

\_intent\_slotsの編集へ  
「first\_number」をクリックします。

employeeNumber

first\_number

second\_number

third\_number

fourth\_number

fifth\_number

sixth\_number

seventh\_number

eighth\_number

ビルトインintent(4)

AMAZON.CancelIntent

AMAZON.HelpIntent

AMAZON.StopIntent

AMAZON.NavigateHomeIntent

スロットタイプ(1)

LIST\_OF\_NUMBER

JSONエディター

インターフェース

エンドポイント

intent履歴

画面表示

スキル内商品

社員番号は {first\_number} {second\_number} {third\_number} {fourth\_number} {fifth\_number} {sixth\_number} {seventh\_number} {eighth\_number} です

社員番号は {first\_number} {second\_number} {third\_number} {fourth\_number} {fifth\_number} {sixth\_number} {seventh\_number} {eighth\_number} 番

社員番号は {first\_number} {second\_number} {third\_number} {fourth\_number} {fifth\_number} {sixth\_number} {seventh\_number} {eighth\_number} 番で

社員番号は {first\_number} {second\_number} {third\_number} {fourth\_number} {fifth\_number} {sixth\_number} {seventh\_number} {eighth\_number} 番です

社員番号は {first\_number} {second\_number} {third\_number} {fourth\_number} {fifth\_number} {sixth\_number} {seventh\_number} {eighth\_number}

< 1 - 5 / 60 > [すべて表示](#)

ダイアログデリゲートのルール

このintentでは、ダイアログ管... > [これが無効にされている理由](#)

intentスロット(8)

順序	名前	スロットタイプ	アクション
1	first_number	LIST_OF_NUMBER	<a href="#">ダイアログを編集</a> <a href="#">削除</a>
2	second_number	LIST_OF_NUMBER	<a href="#">ダイアログを編集</a> <a href="#">削除</a>

[ダイア](#)

## \_intent\_slotを編集

「～このスロットは必須ですか？」のスイッチをオンにした後、Alexaからの問いかけの言葉を「Alexaの音声プロンプト」に、

ユーザーの応答

「{first\_number}」を「ユーザーの発話」にそれぞれ入力して「+」をクリックします。

編集が終わったら、「モデルを保存」しておきます。

The screenshot displays the Alexa Developer Console interface for editing an intent slot. On the left, a sidebar lists various intents and slot types. The 'first\_number' slot under the 'employeeNumber' intent is selected. The main panel shows the 'LIST\_OF\_NUMBER' slot type. A message indicates that auto-delegation is enabled. The 'Dialog' tab is active, showing the 'Slot Input' section. A toggle switch for 'Is this slot required?' is turned on. Below it, the 'Alexa's voice prompt' field contains the text 'ユーザーにこのスロットの入力を求めるために、Alexaが話しかける内容を入力してください'. The 'User's utterance' field contains '{first\_number}'. Red boxes and arrows highlight the toggle switch, the voice prompt text, and the '+' button next to the utterance. The 'Slot Confirmation' section at the bottom has a toggle switch for 'Does this slot require confirmation?' which is currently off.

## インターフェースを無効化してビルド

「オートデリゲート」のスイッチをオフにし、すべてのインターフェースをオフにしたら、「インターフェースを保存」をクリックします。

最後に、「HelloWorldIntent」は使わないので削除し、「モデルをビルド」をクリックします。

以上で、対話モデルの作成は完了です。

The screenshot shows the Alexa Developer Console interface. At the top, there are buttons for '日本語' (Japanese), 'インターフェースを保存' (Save Interfaces), 'バージョン' (Version), 'モデルをビルド' (Build Model), and '発話プロファイラー' (Utterance Profiler). The left sidebar shows the 'カスタム' (Custom) section with '対話モデル' (Conversation Model) selected. Under '呼び出し名' (Invocation Name), there is a list of intents: 'Intent(6)' (expanded), including 'HelloWorldIntent', 'employeeNumber' (expanded), and 'ビルトインIntent(4)' (expanded). The 'employeeNumber' intent has several slots: 'first\_number', 'second\_number', 'third\_number', 'fourth\_number', 'fifth\_number', 'sixth\_number', 'seventh\_number', and 'eighth\_number'. The 'ビルトインIntent(4)' section includes 'AMAZON.CancelIntent', 'AMAZON.HelpIntent', 'AMAZON.StopIntent', and 'AMAZON.NavigateHomeIntent'. The 'スロットタイプ(1)' (Slot Type) section includes 'LIST\_OF\_NUMBER'. The 'JSONエディター' (JSON Editor) is also visible.

The main content area is titled 'インターフェース' (Interfaces). It contains a message: 'インターフェースを有効にすると、対話モデルに必要なIntentを追加しなければならない場合があります。変更を有効にするには、インターフェースの変更を保存し、モデルを再ビルドするという両方の作業を行う必要があります。' (Enabling interfaces may require adding intents necessary for the conversation model. To enable changes, you must save the interface changes and rebuild the model.)

Below the message is a table of interfaces:

インターフェース名	説明	スイッチ
Audio Player	AudioPlayerインターフェースでは、オーディオをストリーミングし、再生状況を監視するディレクティブとリクエストを提供します。AudioPlayerインターフェースの詳細はこちら。	<input type="checkbox"/>
Displayインターフェース	画面付きEchoデバイスでは、画面と音声対話の両方を使用するAlexaスキルを作成できます。Displayインターフェースの詳細はこちら。	<input type="checkbox"/>
VideoApp	VideoAppインターフェースでは、Echo ShowのネイティブビデオファイルをストリーミングするVideoApp.Launchディレクティブを提供します。VideoAppインターフェースの詳細はこちら。	<input type="checkbox"/>
Alexa Presentation Language	Alexa Presentation Language (APL)を使うと、Echo Show、Echo Spot、Fire TV向けのマルチモーダルスキルを開発できます。オーサリングツールとシミュレーターを使って、APLドキュメントを作成、テストしましょう。ベータ版の制約については、 <a href="#">APIリファレンス</a> を参照してください。また、 <a href="#">ブログ (英語)</a> でもこの機能について詳しく説明しています。	<input type="checkbox"/>
オートデリゲート	ダイアログモデルに基づいて、Alexaに自動的にダイアログの各ステップを判断して完成させます。ダイアログが完成すると、単一のIntentRequestを取得します。この設定はIntentレベルで書きできます。オートデリゲ	<input checked="" type="checkbox"/>

Red arrows indicate the workflow: from 'インターフェースを保存' to the 'Audio Player' interface, then to the 'オートデリゲート' switch, and finally to 'モデルをビルド'. A red box highlights the 'オートデリゲート' switch.



# スキルの作成方法③

## AWS Lambda関数

## ソースコードについて

index.jsは、以下の記事のLaunchRequestHandler、regist関数、RegistIntentHandlerを参考に、alexa developer consoleのコードエディタにデフォルトで入っているコードを修正して作りました。

【Alexa】Connpass用の受付スキルを作ってみた #Alexa #AlexaDevs  
(<https://dev.classmethod.jp/voice-assistant/try-to-develop-registration-skill/>)

package.jsonは上記記事のものをそのまま使っています。

また、env.jsというファイルを追加しました。

次頁より、デフォルトのソースコードからの修正点を解説します。

index.jsについては、解説の内容のほか、HelpIntentHandler、CancelAndStopIntentHandler、IntentReflectorHandler、ErrorHandlerのspeechTextを日本語にする修正もしています。

# LaunchRequestHandler

index.jsの解説をします。呼び出し名単体で呼び出したときの、LaunchRequestHandlerのhandleメソッドは、以下のようにしました。

```
handle(handlerInput) {  
    console.log("LaunchRequestHandlerのhandleの中");  
    return handlerInput.responseBuilder  
        .addDelegateDirective({  
            name: 'employeeNumber',  
            confirmationStatus: 'NONE',  
            slots: {}  
        })  
        .getResponse();  
}
```

speak関数、reprompt関数を削除して、first\_numberスロットに設定したAlexaの音声プロンプトをそのまま読み上げさせることにしました。

## requestAppsScript関数

Google Apps Scriptにリクエストを送る関数をrequestAppsScript関数として以下のようにしました。

```
const request = require('request-promise');
const requestAppsScript = employeeNum => {
  console.log("requestAppsScript関数の中");
  const id = require('./env').scriptID;
  const url = "https://script.google.com/macros/s/" + id +
    "/exec?ticketnumber=" + employeeNum;
  console.log("url:", url);
  return request(url);
};
```

社員番号を受け取ってGoogle Apps Scriptにリクエストを送ります。

URLの一部はenv.jsから読み込みます。

env.jsのサンプル(sample\_of\_env.js)

```
const scriptID = "xxxxxxxxxxxxxxxxxxxxxx";
module.exports.scriptID = scriptID;
```

## EmployeeNumberHandler①

employeeNumberIntentに対応したEmployeeNumberHandlerのhandleは以下のようにしました。

```
async handle(handlerInput) {  
    console.log("employeeNumberHandlerのhandleの中");  
    let speechText = "";  
    let response = handlerInput.responseBuilder;  
    const spokenNumbers = [];  
    try {  
        spokenNumbers.push(Alexa.getSlotValue(handlerInput.requestEnvelope,  
            "first_number"));  
        spokenNumbers.push(Alexa.getSlotValue(handlerInput.requestEnvelope,  
            "second_number"));  
        -----省略-----  
    }  
}
```

first\_numberから、eighth\_numberまでのスロット値を配列に格納します。

## EmployeeNumberHandler②

取得したスロット値を連結させ、replaceWithArabicNumerals関数で処理します。  
「NaN」は処理できないので、除外します。

```
let employeeNum = "";
for (let i = 0; i < 8; i++) {
  console.log("数字:", spokenNumbers[i]);
  if (spokenNumbers[i]) {
    employeeNum += spokenNumbers[i];
  }
}
if (!Number.isNaN(employeeNum)) {
  employeeNum = replaceWithArabicNumerals(employeeNum);
}
console.log("社員番号:", employeeNum);
```

-----続く-----

## replaceWithArabicNumerals関数

Alexaが認識した数字は、漢数字としてスロットに入ります。

そこでこの関数を使い、〇～九までの漢数字をすべて、0～9までのアラビア数字に置換します。

```
const replaceWithArabicNumerals = numString => {  
  const arabicNumerals = numString.replace(/〇/g, "0")  
    .replace(/一/g, "1")  
    .replace(/二/g, "2")  
    .replace(/三/g, "3")  
    .replace(/四/g, "4")  
    .replace(/五/g, "5")  
    .replace(/六/g, "6")  
    .replace(/七/g, "7")  
    .replace(/八/g, "8")  
    .replace(/九/g, "9");  
  return arabicNumerals;  
};
```

## EmployeeNumberHandler③

取得した社員番号が数字のみだった場合に、非同期でrequestAppsScript関数を呼び出し、戻り値をAlexaが読み上げるspeechTextに渡します。

```
if (isNaN(employeeNum)) {  
    speechText = "数字でない";  
} else {  
    await requestAppsScript(employeeNum)  
        .then(val => {  
            console.log("speechText: " + val);  
            speechText = val;  
        })  
        .catch(e => console.error(e));  
}
```

-----続く-----



## EmployeeNumberHandler④

requestAppsScript関数から"missing"が返ってきたとき(該当する社員番号がなかったとき)、speechTextに新しいテキストを渡し、addElicitSlotDirective関数でスロットの入力を促します(※1)。

```
if (speechText === "missing") {  
  console.log("値が取れなかった");  
  if (/¥?/.test(employeeNum)) {  
    speechText = "よくわかりませんでした。受付を始めからやり直してください。";  
  } else {  
    speechText = "<say-as interpret-as='digits'>" + employeeNum  
      + "</say-as>番の社員番号が見つかりません。もう一度言ってもらえますか?";  
  }  
  response.reprompt(speechText).addElicitSlotDirective(  
    "first_number", "second_number", "third_number", "fourth_number",  
    "fifth_number", "sixth_number", "seventh_number", "eighth_number");  
}
```

-----続く-----

## EmployeeNumberHandler⑤

誤認識・言い間違いで数字以外の値が\_intentに入ってきたときにも、同様の処理をします。

```
    } else if (speechText === "数字でない") {  
        speechText = "よくわかりませんでした。もう一度言ってもらえますか？";  
        response.reprompt(speechText).addElicitSlotDirective(  
            "first_number", "second_number", "third_number", "fourth_number",  
            "fifth_number", "sixth_number", "seventh_number", "eighth_number");  
    }  
  
    } catch (e) {  
        console.error(e);  
    }  
    return response.speak(speechText)  
        .reprompt("よくわかりませんでした。もう一度言ってもらえますか？")  
        .getResponse();  
}
```

最後に、speak関数でspeechTextの値をAlexaが読み上げた後、社員番号を取得できなかったとき、reprompt関数で指定したテキストをAlexaが読み上げます。reprompt関数を使うことで、セッションが継続され、続けて受付できるようになっています([※2](#))。

## EmployeeNumberHandler⑥

requestAppsScript関数、replaceWithArabicNumerals関数、EmployeeNumberHandlerを追加したら、LambdaハンドラーのaddRequestHandlers関数の引数に追加します。

デバッグ用のIntentReflectorHandlerは、使わないのでコメントアウトしました。

```
exports.handler = Alexa.SkillBuilders.custom()
    .addRequestHandlers(
        LaunchRequestHandler,
        EmployeeNumberHandler,
        HelpIntentHandler,
        CancelAndStopIntentHandler,
        SessionEndedRequestHandler,
        //IntentReflectorHandler, // make sure IntentReflectorHandler is last so it
        //doesn't override your custom intent handlers
    )
    .addErrorHandlers(
        ErrorHandler)
    .lambda();
```

index.jsの解説は以上です。

package.json

package.jsonの"dependencies"のうち、"ask-sdk-core"と"ask-sdk-model"のバージョンを以下のように変更します。

また、"request"と"request-promise"を追加します。

```
"dependencies": {  
  "ask-sdk-core": "^2.5.1",  
  "ask-sdk-model": "^1.9.0",  
  "aws-sdk": "^2.326.0",  
  "request": "^2.88.0",  
  "request-promise": "^4.2.4"  
}
```

Lambda関数のソースコードの解説は以上です。

## コードエディタへ

前頁までのソースコードを適用します。

ソースコードを編集するために、「コードエディタ」をクリックします。

The screenshot shows the Alexa Developer Console interface. The top navigation bar includes links for Skills, Intent, Build, **コードエディタ** (Code Editor), Test, Publish, Confirm, and Report. The left sidebar shows the 'Custom' section with 'Dialog Model' selected. Under 'Dialog Model', 'Intent (5)' is expanded, showing a list of intents: 'employeeNumber' (selected), 'first\_number', 'second\_number', 'third\_number', 'fourth\_number', 'fifth\_number', 'sixth\_number', 'seventh\_number', and 'eighth\_number'. Below this, 'Built-in Intent (4)' lists 'AMAZON.CancelIntent', 'AMAZON.HelpIntent', 'AMAZON.StopIntent', and 'AMAZON.NavigateHomeIntent'. At the bottom, 'Slot Type (1)' shows 'LIST\_OF\_NUMBER'. The main content area is titled 'Intent / employeeNumber'. It displays 'Sample utterances (60)' with a list of sample utterances for the 'employeeNumber' intent, each containing a list of numbers (e.g., '社員番号は {first\_number} {second\_number} {third\_number} {fourth\_number} {fifth\_number} {sixth\_number} {seventh\_number} {eighth\_number} です'). Below the list, there is a section for 'Dialog Delegate Rules' with a dropdown menu set to 'Skill settings fallback'.

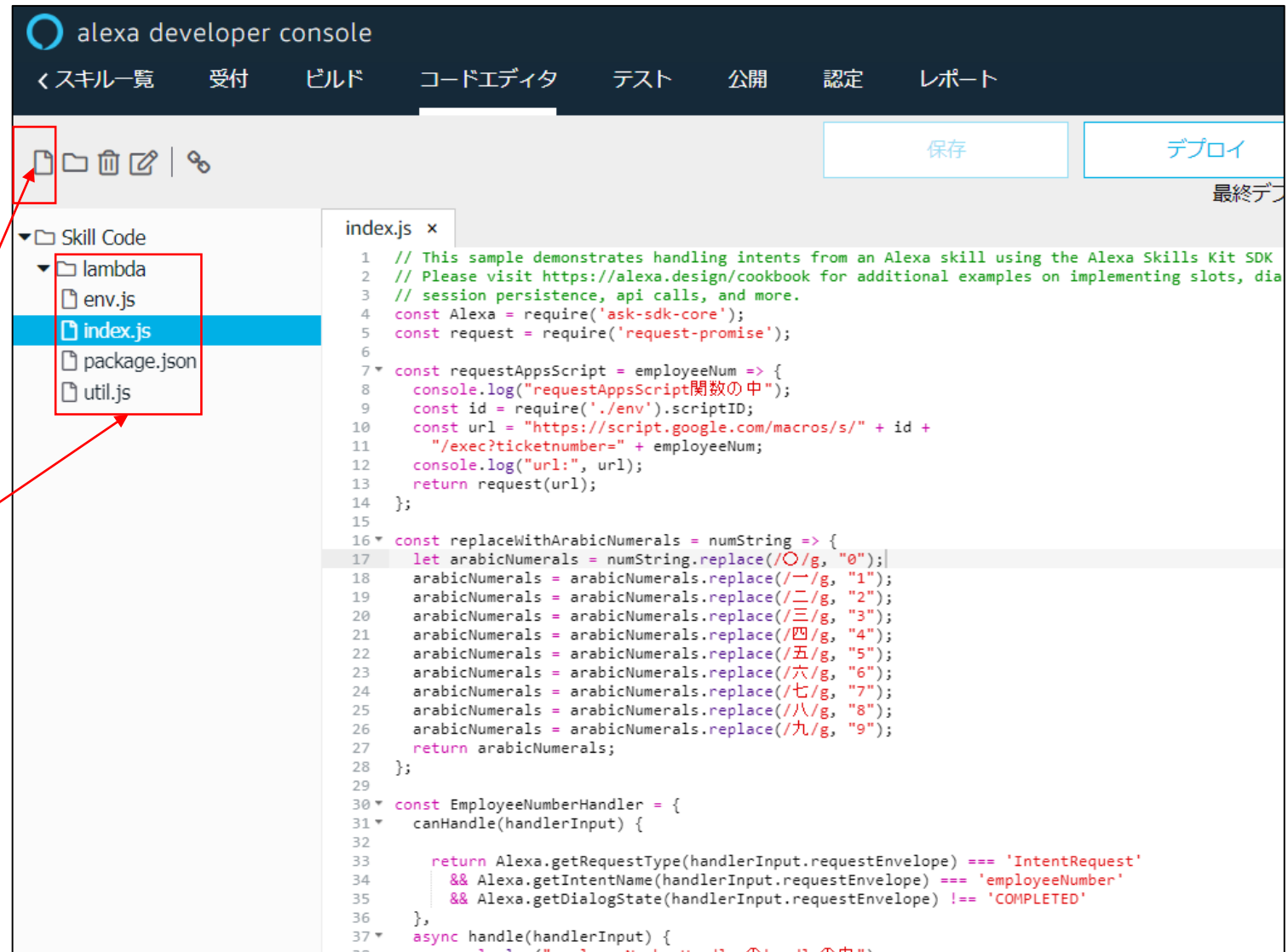
## ソースコードを編集

ファイル名をダブルクリックすると、そのファイルを編集できます。新しいファイルは、左上の「ファイルを作成」ボタンから作成します。

lambdaフォルダの直下に index.js、env.js、package.jsonを配置します。

ソースコードは以下の場所にあります。

<https://github.com/forshoes-admin/Alexa-reception>



# デプロイ

編集中のファイルは「保存」ボタンまたは[Ctrl]+[S]で保存できます。

編集が終わったら、「デプロイ」ボタンをクリックしてデプロイします。

以上で受付スキルの完成です。  
本稿で挙げた記事も参考に、  
Alexaとのやり取りで使う言葉や、  
社員番号(その他ID等)の桁数を変えて使ってみるのも良いと思います。



## 参考

【Alexa】Connpass用の受付スキルを作ってみた #Alexa #AlexaDevs  
(<https://dev.classmethod.jp/voice-assistant/try-to-develop-registration-skill/>)

[日本語Alexa] 4桁の数字を確実に受けとるためには ～より自然に会話できるスキル作成のために～

(<https://dev.classmethod.jp/cloud/4-digit-number-custom-slot/>)

※1: インテントチェーン(Intent Chaining) スキル側で自由に別のインテントに遷移できるようになったので、しびれるぐらい自然に会話が進むようになった(新機能)

(<https://dev.classmethod.jp/cloud/alexa-intent-chaining/>)

※2: [日本語Alexa] Alexa SDK for Node.js Ver2入門(その3)レスポンスの作成

(<https://dev.classmethod.jp/cloud/alexa-sdk-v2-third/>)