

Google Apps Script と Slack Incoming Webhooks を使った Slack bot 入門

@hitsumabushi845

目次

1	はじめに	3
1.1	対象読者	3
2	概要	4
2.1	Google Apps Script とは	4
2.2	Incoming Webhooks とは	7
2.2.1	Custom Integration(Legacy)	8
2.2.2	App(Recommended)	11
3	単独で動作する Google Apps Script	17
3.1	Example: GAS から Incoming Webhooks を使って Slack に投稿してみよう	17
3.1.1	スクリプトの実行方法とログイン	17
3.1.2	UrlFetchApp と Web API への HTTP リクエスト	19
3.2	いろいろな Payload の作成と実行例	21
3.2.1	mrkdwn 記法	21
3.2.2	メンション	22
3.2.3	リンク	23
3.2.4	Block Kit	23
3.2.5	Attachments	25
4	Google Forms と GAS	26
4.1	FormApp	26
4.1.1	FormApp を使ったフォーム作成	27
4.2	Google Apps に紐づくスクリプト	30
4.2.1	イベントトリガ	31
4.2.2	Event Object	33
4.3	質問形式と取り出され方	35
4.4	Practice: Google Forms × Slack	38

5	Google Spreadsheets と GAS	39
5.1	Spreadsheet の構造 - プログラムでスプレッドシートを扱うとはどういうことか	39
5.2	SpreadsheetApp	40
5.2.1	スプレッドシートを取得する	41
5.2.2	スプレッドシートからシートを取得する	42
5.2.3	シートから指定した範囲のセル/値を取得する	42
5.3	Practice: Google Spreadsheets × Slack	45
5.3.1	時間主導型トリガ	45
6	Advanced Practice: Google Forms × Google Spreadsheets × Slack	47
7	おわりに	47
付録 A	実装例	48
A.1	Practice: Google Forms × Slack	48
A.2	Practice: Google Spreadsheets × Slack	48
付録 B	更新履歴	49

1 はじめに

こんにちは。hitsumabushi845 です。2019 年冬に私の勤務している株式会社 Works Human Intelligence にて全 4 回ほど実施していた Google Apps Script × Slack 講座の内容を大幅な加筆とともにテキストにしました。

この講座はもともとスライドを使ったハンズオン形式で実施していましたが、テキストにするにあたり改めて構成を整理しなおし、ハンズオンで行っていた部分は練習問題として各章の最後に掲載しています。練習問題以外の部分でも少しずつ写経ができるようにし、実際に手を動かして確認できるよう意識しています。なお、本文中の赤文字になっているところは、対応する Web サイトへのリンクが埋め込まれています。必要に応じて参照してください。

1.1 対象読者

主に以下を対象としています。Google Apps Script は JavaScript 1.6 ベースのスクリプト言語です^{*1}が、その文法については本テキストでは触れません。しかし、本テキストでは基本的なデータ型、条件分岐、ループ程度しか扱いませんので、自信のない方は [JavaScript Primer](#) を参照することをおすすめします。

- プログラミング初級者程度の知識があり、なにか作ってみたい人
- Slack bot を作ってみたい人
- Google Apps と Slack の連携を強化したい人
- Google Apps Script に興味がある人

^{*1} 2020 年 2 月から V8 ランタイムが選べるようになりました。

2 概要

本章では、このテキストで取り扱う2つのツール、Google Apps Script と Slack Incoming Webhooks について説明します。Google Apps Script, Incoming Webhooks とはなにか、どういったことができるのか、そしてそれらの使い方といった内容を説明します。これらの具体的な使用方法については次章以降で説明します。

2.1 Google Apps Script とは

Google Apps Script (GAS) は、G Suite 上にある軽量スクリプトプラットフォーム (FaaS) です。**G Suite 上にある**というのがポイントで、Google Spreadsheets や Google Docs, Gmail などといった Google Apps との親和性が高いことが特徴です。スクリプトから Google Apps 上のデータを操作したり、Google Apps からスクリプトを実行したりと、Google Apps とスクリプトをシームレスに連携することができます。

一方で、クラウドサービスとしてのスクリプト実行環境であるため、スクリプトの呼び出し回数や API の実行時間に**制限があります**。したがって、いわゆるバッチ処理などといった、時間のかかる処理を GAS 上で行うのは現実的ではありません。

GAS の画面には、<https://script.google.com/home> からアクセスできます (図 1)。



図 1 Google Apps Script メイン画面

画面左上にある「新しいプロジェクト」ボタンをクリックすると、図 2 に示すようなエディタ画面が表示されます。Google Apps Script はこの画面上でプログラミングを行います^{*2}。

^{*2} ローカルでの開発を可能にする **Clasp** というツールもありますが、本テキストでは扱いません。

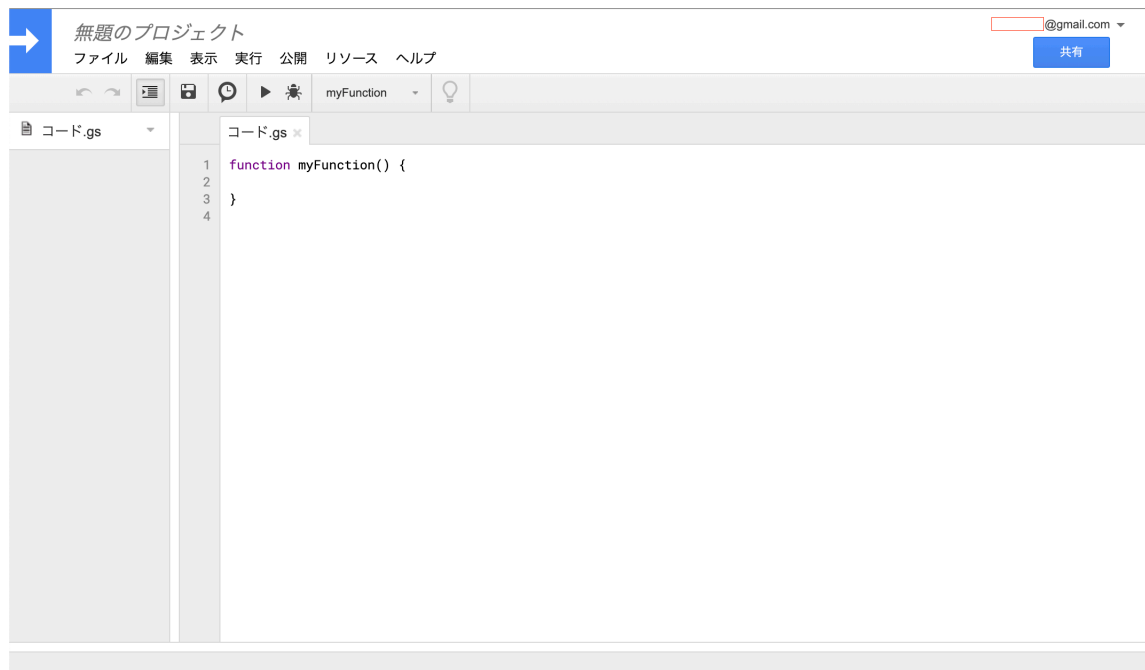


図2 Google Apps Script エディタ画面

加えて、なぜ Google Apps Script 単体での HOW-TO ではなく、Slack との連携を主題に置いているか、といった点にも触れておきたいと思います。GAS は G Suite を活用する上で非常に強力なツールですが、GAS 以外の手段については意識したほうがよいでしょう。というのも、各 Google Apps それ自体にも多くの機能が備わっており、スクリプトを書くまでもなく Google Apps 自体の機能で目的を実現できる場合もあるからです。

例として、「Google Spreadsheets に入力された日本語の文章を英語に翻訳する」といったケースを考えてみます。先程説明したように、GAS は各種 Google Apps とシームレスに連携することができ、GAS には Google Spreadsheets と Google Translate への API も含まれています。したがって、

1. Google Spreadsheets に入力されている日本語の文章を取得する
2. 取得された文章を Google Translate に流し込む
3. Google Translate によって翻訳された文章を Google Spreadsheets に書き込む

といった操作は GAS を用いて以下のようなスクリプトを書くことで実現できます。

```
1 function translateSample() {  
2  
3   var spreadsheet = SpreadsheetApp.openById('your/spreadsheet/id');  
4   var sheet = spreadsheet.getSheetByName('your/sheet/name');  
5   var ja_contents = sheet.getRange(2,1,sheet.getLastRow()-1).getValues();  
6   var translated_text = ""  
7  
8   for (var i = 0; i < ja_contents.length; i++) {  
9     translated_text = LanguageApp.translate(ja_contents[i], 'ja', 'en');  
10    sheet.getRange(i+2, 2).setValue(translated_text);  
11  }  
12 }
```

さて、このようなケースで GAS を用いるのは本当にベストな選択でしょうか？実は、Google Spreadsheets には **GOOGLETRANSLATE** という関数が用意されています。したがって、そもそもスクリプトを書く必要すらなく、Google Spreadsheets の持つ機能だけで行いたかったことは実現できます^{*3}。

上記の例では Google Spreadsheets と Google Translate の連携が Google Spreadsheets だけで可能であることを説明しました。しかし、一般的には複数の Google Apps を組み合わせた処理はそれほど機能として用意されていません。なので、**複数の Google Apps を連携させる用途**、**Google Apps と外部サービスの API を連携させる用途**などでは GAS が効果を発揮する可能性が高いです。そのため、本テキストでも **Slack との連携**を主軸に置いています。

^{*3} “翻訳の質”という観点で考えると、GAS を用いた方法のほうが良い場合もあります。実際には実装コストと併せて考える必要があると思います。see: <https://qiita.com/kazinoue/items/82779a11af8a2dea0d21>

2.2 Incoming Webhooks とは

Incoming Webhooks はアプリケーションやプログラムから Slack にメッセージを投稿する方法 (API) のひとつです。Webhook URL と呼ばれる URL に、投稿したい内容を持った HTTP リクエストを送ると、Slack にメッセージが投稿されます。

API とは

API;Application Programming Interface とは、以下のように説明されます。

API とは、あるコンピュータプログラム (ソフトウェア) の機能や管理するデータなどを、外部の他のプログラムから呼び出して利用するための手順やデータ形式などを定めた規約のこと。

上記の記述は抽象的なので、今回のテキストの内容に即して言葉を置き換えてみると、“*Incoming Webhooks* とは、*Slack* の機能や管理するデータなどを、*GAS* をはじめとしたプログラムから呼び出して利用するための手順や規約” と考えることができます。

一方で、*GAS* 自体も API と呼ぶことができます。この場合、「あるコンピュータプログラム」は Google Apps, 「外部の他のプログラム」を *GAS* と置き換えます。先程のスクリプトの例で登場した **SpreadsheetApp** や **LanguageApp** はそれぞれ、Google Spreadsheets, Google Translate を操作するためのオブジェクトですので、これらは API となります。

特に、エンドポイント URL への HTTP リクエストによって扱う API のことを、**Web API** と呼びます。したがって、Incoming Webhooks は Web API に該当します。

さて、Incoming Webhooks では“投稿したい内容”を HTTP リクエストとして渡すと Slack にメッセージを投稿できるのでした。この“投稿したい内容”として必要な情報には何があるでしょうか？少なくとも、**投稿する内容 (本文)** と **投稿する先 (チャンネルや DM 先)** は必要そうです。実は、Incoming Webhooks では、図 3 のように、通常私たちが Slack に投稿する形式以上の表現が可能です。そのため、Incoming Webhooks を使ってメッセージを投稿する際に指定できる情報は**本文**と**チャンネル**だけでなく、もっとたくさんありそうです。

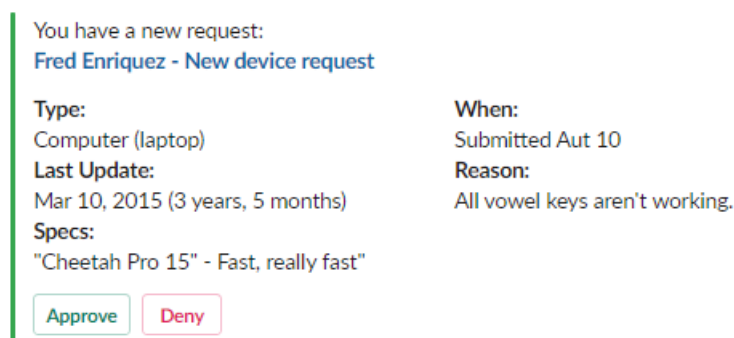


図 3 Incoming Webhooks による投稿例

現在、Slack には以下の 2 つの形式の Incoming Webhooks があります。

1. Custom Integration

カスタム インテグレーションとして設定する Incoming Webhooks。こちらでしか使えない機能もある

が、現在では非推奨の形式。

2. App

Slack App として設定する Incoming Webhooks。現在はこちらが推奨されている。

それぞれの形式の Incoming Webhooks について詳しく説明します。

2.2.1 Custom Integration(Legacy)

Slack では、よく使いそうな機能が組み込みのアプリケーションとして用意されており、必要に応じて用途に即した形で設定を追加できます。これを**カスタムインテグレーション**といいます。**よく使いそうな機能**には、Incoming Webhooks のほかに Outgoing Webhooks や Slash Commands などがあります。

設定済みのカスタムインテグレーションは、

<https://{ワークスペース名}.slack.com/apps/manage/custom-integrations>
から確認できます。

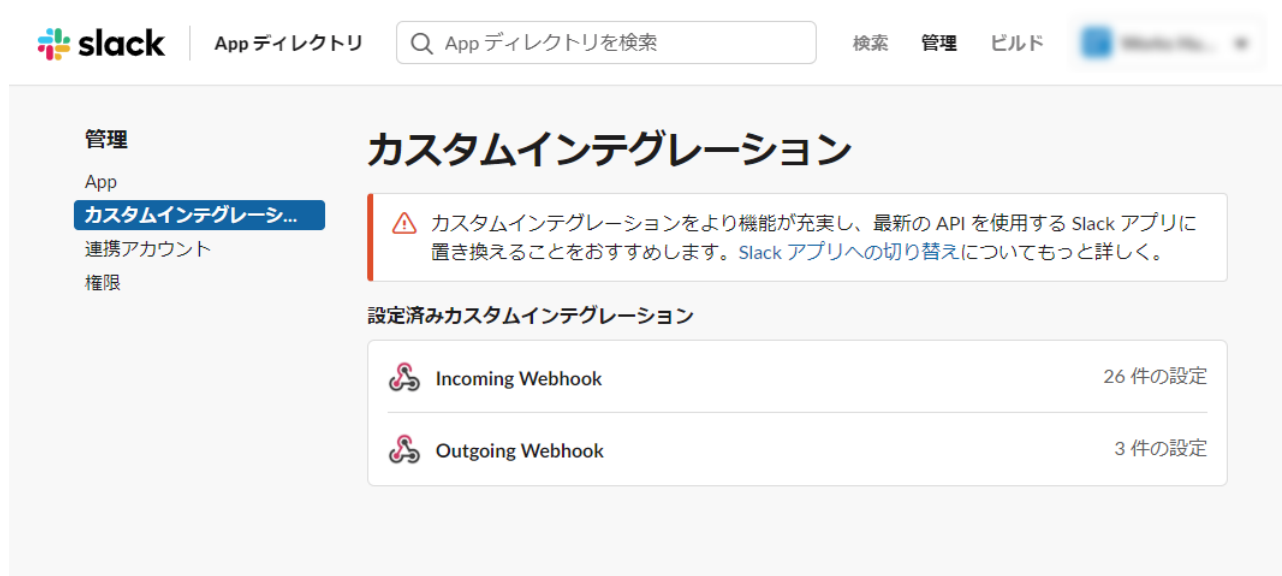


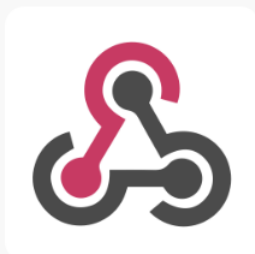
図4 カスタムインテグレーション画面 1

カスタムインテグレーションは、後述する理由により現在では**非推奨**の形式ですので、上記の画面上にも**Slack アプリに置き換えることをおすすめします。**と記載されています。

Incoming Webhooks の設定を追加する場合は、この画面上で Incoming Webhooks と書かれている部分を選択します。

画面上に**設定をリクエストする**といったボタンが表示されています。これをクリックすると、Slack ワークスペースの管理者に設定の追加をリクエストすることができます。ちなみに、このボタンが**設定を追加**になっている場合は、管理者の承認を必要とせず、自由に設定を追加できます。

カスタムインテグレーションの Incoming Webhooks に**デフォルト**のチャンネルやユーザー名・アイコンなどを決めた設定を追加すると、独自の URL が発行されます。この URL が Webhook URL です。この URL に対して、投稿したい内容を持った HTTP リクエストを送ると、事前に設定したデフォルトのチャンネルに対し

[アプリを検索する](#)[設定をリクエストする](#)[アプリのヘルプ](#)[お支払い条件](#)

このアプリの不適切なコンテンツや動作については Slack に報告してください。

Incoming Webhook

[アプリ情報](#)[設定](#)

このアプリは Slack が開発しました。

このアプリは、サードパーティーのサービスと Slack を連携することができるように Slack チームのメンバーによって作成されました。このようなアプリは、Slack Enterprise Grid や Slack for Teams といった当社のコア製品のサポートのように、Slack によるテスト、記録、またはサポートが行われていない場合があります。feedback@slack.com 宛てにこれらアプリのフィードバックを送ることができます。

使用するデータは、Slack がすでにアクセスできるデータのみです (詳しくは、[Slack プライバシーポリシー](#)をご覧ください)。このアプリを有効化または/および使用することで、Slack の一部ではないサービスに接続することになる可能性があります。

設定



Webhook URL: [https://hooks.slack.com/services/T000000000000000000/B000000000000000000](#)



Webhook URL: [https://hooks.slack.com/services/T000000000000000000/B000000000000000000](#)

図 5 カスタムインテグレーション画面 2

て、デフォルトのユーザー名・アイコンで投稿が行われます。

なお、このデフォルトのチャンネルやユーザー名、アイコンはいつでも変更できます。

カスタムインテグレーションの Incoming Webhooks でのみ可能な使い方として、**投稿先のチャンネル、投稿のユーザー名、アイコンを自由に変えられる**ことが挙げられます。

先程説明したように、投稿する内容のみを使って Webhook URL に対して POST を行うと、デフォルトのチャンネルに対して投稿が行われます。例として、以下の Curl コマンドの実行例を見てみます。

```
1 > curl -X POST --data-urlencode "payload={\"text\": \"TEST\" }" https://hooks.slack.com/services/your/webhook/url
2 ok
```

ソースコード 2 Curl sample

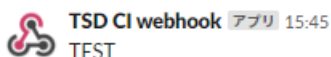


図 6 Incoming Webhooks 実行例

図 6 では、事前に設定したデフォルトのユーザー名 (TSD CI webhook) で投稿されています。アイコンは設定していないため、カスタムインテグレーションのアイコンが表示されています。

ここで、Webhook URL に送信する情報 (上記 Curl コマンドの payload 要素) をいくつか追加してみましょう。

```
1 > curl -X POST --data-urlencode "payload={\"text\": \"TEST\", \"channel\": \"@shimada_ken\", \"username\": \"This is custom integration\", \"icon_emoji\": \":tada:\" }" https://hooks.slack.com/services/your/webhook/url
2 ok
```

ソースコード 3 Curl sample2

payload 要素が長くなりすこし見づらくなってしまいました。payload の部分だけ見やすく整形すると以下のようになります。

```
1 payload = {
2   "text": "TEST",
3   "channel": "@shimada_ken",
4   "username": "This is custom integration",
5   "icon_emoji": ":tada:"
6 }
```

ソースコード 4 Formed payload

ここでは、text の他に、channel, username, icon_emoji を設定しています。それぞれ、送信先のチャンネル、表示名、アイコンとなる絵文字に対応します。上記 Curl コマンドを実行すると、以下のような内容が私にダイレクトメッセージで送信されます。ユーザー名が TSD CI webhook から This is custom integration に、アイコンがクラッカーの絵文字になっていることがわかります。カスタムインテグレーションの Incoming Webhooks では、アイコンを Slack の絵文字で代替することができます。もちろん、画像を指定することも可能です。

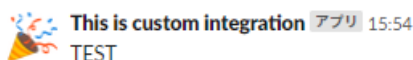


図 7 Incoming Webhooks 実行例 2

このように、送信先やユーザー名、アイコンを自由に変えられるのが、カスタムインテグレーションの Incoming Webhooks の特徴です。そのため、**1 つの Webhook URL で様々なプログラム・アプリケーションに対応できます。**

2.2.2 App(Recommended)

一方で、1 つの Webhook URL には 1 つの役割しか与えないほうがよい、という考えがあります。なぜなら、あるカスタムインテグレーションによって作成された Webhook URL があつたとき、これがどのプログラム・アプリケーションに使われているのか把握できないといった問題が発生するからです。カスタムインテグレーションの Webhook URL は、任意のチャンネルに対して、任意の見た目（ユーザー名やアイコン）で投稿ができてしまうので、極端な話をするとこれは 1 つのワークスペースに 1 つあればよいということになってしまいます。つまり、カスタムインテグレーションによる Incoming Webhooks では、Webhook URL と実際に URL が使用されるアプリケーション間の関係といった情報が欠落してしまっているのです。

そこで、Slack App の一部としての Incoming Webhooks が登場します。これは、Slack App としてワークスペースに設定するので、その App の役割に即して使われる想定で存在します。当然、App の役割は決まっており、ユーザー名やアイコン、投稿するチャンネルが変わることはないはずです。ですので、この Incoming Webhooks ではこれらを変えることができません。1 つの Webhook URL には必ず 1 つのチャンネルが結びついており、例外はありません。もし 1 つの App が複数のチャンネルに投稿しなければならない場合は、App から Webhook URL を**投稿先の数だけ**発行します。

App として Incoming Webhooks を利用する手順は以下の通りとなります。

1. Slack App の作成
2. Incoming Webhooks の有効化
3. Incoming Webhooks の作成

まず、Slack App を作成します。これは https://api.slack.com/apps?new_app=1 から作成できます。この URL にアクセスすると、以下の図 8 のような画面が表示されます。

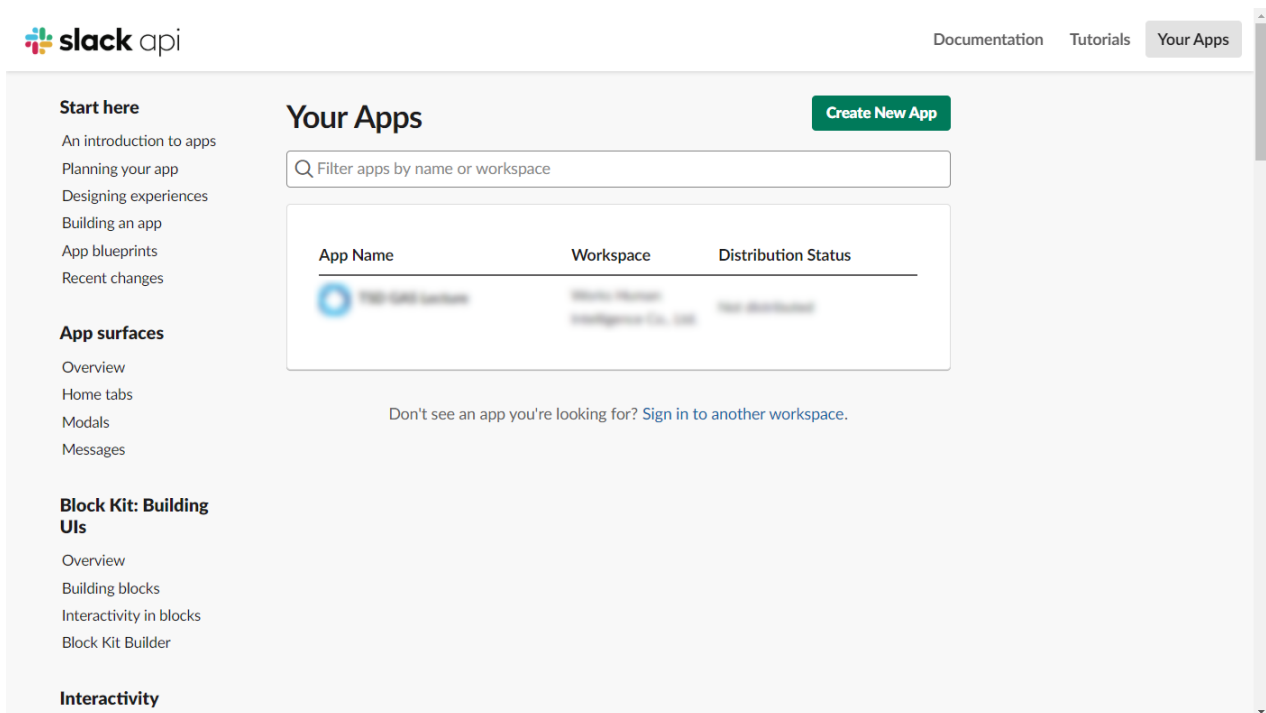


図 8 App 作成画面

この画面右上の“Create New App”というボタンをクリックすると、図 9 のダイアログが表示されます。ここに、作成する App 名と、App を設定するワークスペースを入力し、“Create App” ボタンをクリックします。ワークスペースのポリシーにより、App の設定に管理者の承認が必要な場合は、このタイミングで管理者にリクエストが送られます。特に承認が不要な場合は、そのまま App が作成されます。

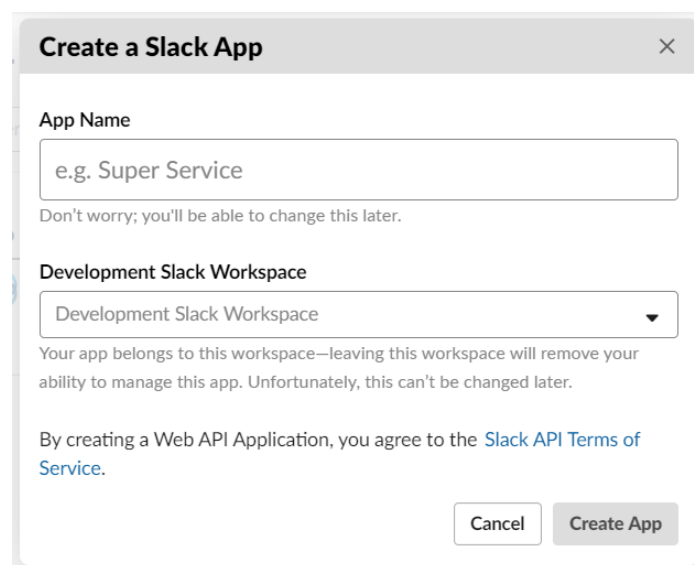


図 9 App 作成画面 2

App が作成されると、App ごとに図 10 のような画面を見ることができます。まず、この画面下部の“**Display Information**” から、この App が Slack に投稿する際のユーザー名とアイコンを設定します。Incoming Webhooks で投稿する際もここで設定したユーザー名とアイコンが使用されます。

App name の部分に設定したいユーザー名を入力して、ユーザー名を設定します。次に、アイコンの部分をクリックし、画像をアップロードしてアイコンを設定します。なお、ここで設定する画像は 512px x 512px から 2000px x 2000px に収まる正方形の画像である必要があります。ちなみに、カスタムインテグレーションの場合は Emoji をアイコンに設定できましたが、App では画像のみとなります。

設定後、画面下部の **Save Changes** をクリックして設定を反映します。

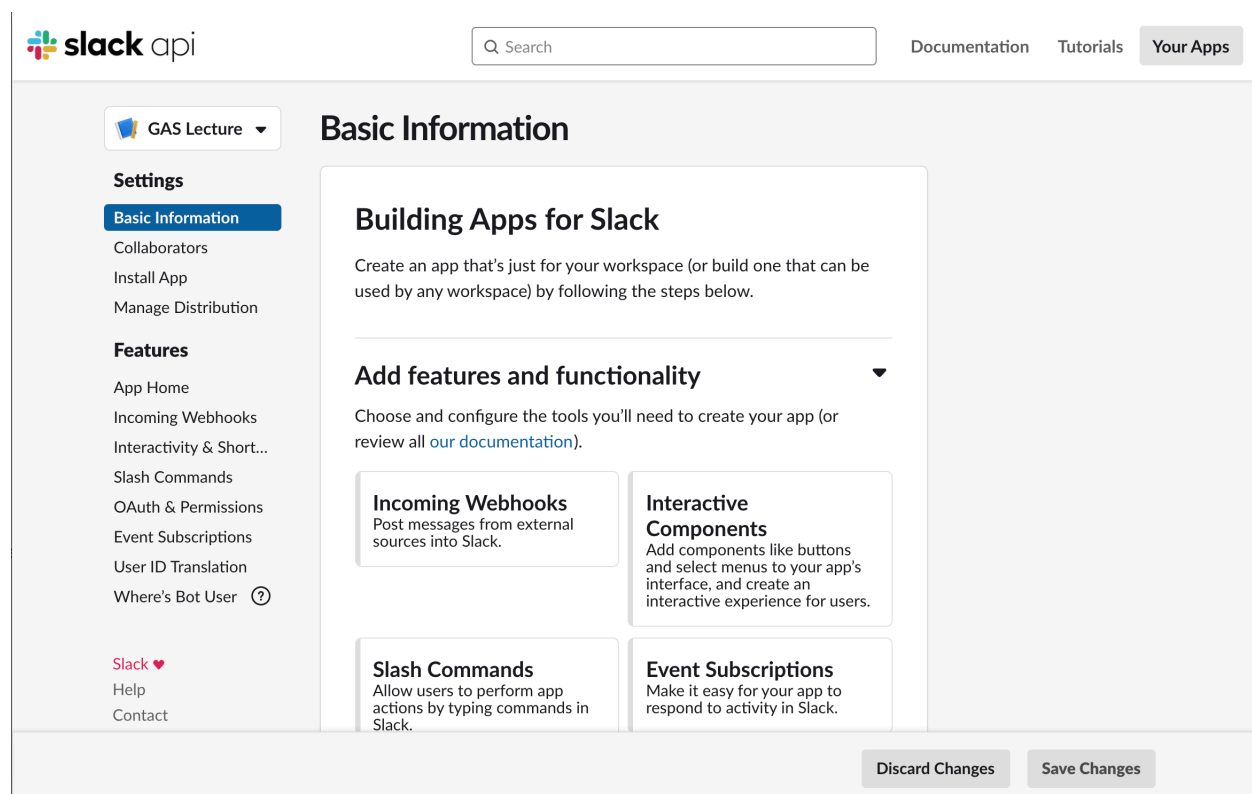



図 10 App 画面



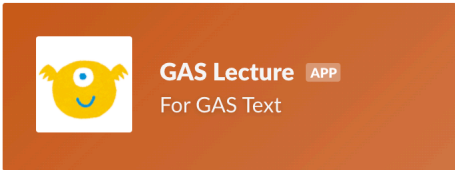
[Documentation](#) [Tutorials](#)

Display Information

This information will be shown in the Slack App Directory and in the Slack App
For more information, view our [App Detail Guidelines](#).

App name	Short description
<input type="text" value="GAS Lecture"/>	<input type="text" value="For GAS Text"/>

App icon & Preview



Background color

Delete App

If your app is listed in the App Directory, please communicate any plans

Discard Changes

Save Changes

図 11 App 画面, Display Information 部分

引き続き, Incoming Webhooks の設定を行っていきましょう。この画面左カラムに, “Incoming Webhooks” というメニューがありますので, こちらを選択します。“Incoming Webhooks” を選択すると, 図 13 のような画面が開きます。はじめは, 画面右上のスイッチが **Off** になっていますので, ここをクリックし, Incoming Webhooks を有効にします。Incoming Webhooks が有効になると, 図 13 の画面になります。画面下部の “Add New Webhook to Workspace”(Webhook の追加に管理者の承認が必要な場合は “Request to Add New Webhook”) をクリックして, Webhook URL を発行します。

GAS Lecture ▼

Settings

Basic Information
Collaborators
Install App
Manage Distribution

Features

App Home
Incoming Webhooks
Interactivity & Shortc...
Slash Commands
OAuth & Permissions
Event Subscriptions
User ID Translation
Where's Bot User ?

Slack ♥

Help
Contact
Policies
Our Blog

Incoming Webhooks

Activate Incoming Webhooks

On ☒

Incoming webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details. You can include [message attachments](#) to display richly-formatted messages.

Adding incoming webhooks requires a bot user. If your app doesn't have a [bot user](#), we'll add one for you.

Each time your app is installed, a new Webhook URL will be generated.

If you deactivate incoming webhooks, new Webhook URLs will not be generated when your app is installed to your team. If you'd like to remove access to existing Webhook URLs, you will need to [Revoke All OAuth Tokens](#).

Webhook URLs for Your Workspace

To dispatch messages with your webhook URL, send your [message](#) in JSON as the body of an `application/json` POST request.

Add this webhook to your workspace below to activate this curl example.

Sample curl request to post to a channel:

```
curl -X POST -H 'Content-type: application/json' --data  
'{"text":"Hello, World!"}' YOUR_WEBHOOK_URL_HERE
```

Webhook URL**Channel****Added By**

図 12 Incoming Webhooks 画面 (App)

“Add New Webhook to Workspace”をクリックすると、投稿するチャンネル名を指定する画面に遷移します。投稿するチャンネル名を選択し、“Install”（管理者の承認が必要な場合は“許可する”）ボタンをクリック

します。



図 13 Incoming Webhooks 画面 (App)

すると、図 13 の画面に、設定したチャンネルへ投稿するための Webhook URL が表示されます。GAS から Slack へ投稿するために、この URL を控えておきます。なお、この URL は App 画面でいつでも確認できます。

3 単独で動作する Google Apps Script

前章では、Google Apps Script を利用する画面へのアクセス方法と、Incoming Webhooks の Webhook URL 発行方法について説明しました。ここからは、実際に Google Apps Script と Incoming Webhooks を使ってみましょう。といっても、この章では Google Apps との連携は行いません。まずは単体の Google Apps Script から Incoming Webhooks を使って Slack にメッセージを投稿してみましょう。

なお、本章以降で使用する Webhook URL はカスタムインテグレーションから発行したものではなく、App から発行したものを使用します。

■本章で扱うトピック

- Google Apps Script の実行方法
- ロガー
- オブジェクトと JSON
- UrlFetchApp

3.1 Example: GAS から Incoming Webhooks を使って Slack に投稿してみよう

本節では、Google Apps Script から Incoming Webhooks の Webhook URL に対して POST リクエストを行い、Slack にメッセージを送信します。[2.1](#) にて紹介した経路で、Google Apps Script のプロジェクトを作成します。

プロジェクトを作成すると、以下のようなスクリプトコードが既に入力された状態でエディタ画面が開きます。

```
1 function myFunction() {  
2  
3 }
```

ソースコード 5 Default Script

`myFunction` という空の関数が定義されている状態です。関数名が `myFunction` では何をする関数かわかりづらいので、この関数名を変えておきましょう。本テキストでは、`slackSender` という名前をつけますが、好きな名前をつけてよいです。

```
1 function slackSender() {  
2  
3 }
```

ソースコード 6 Change function name

3.1.1 スクリプトの実行方法とロギング

まずは GAS の実行方法を紹介します。ですが、まだ `slackSender` 関数にはなにも処理を書いていないので、当然このまま実行するとなんの処理も実行されません。なので、実行できていることを確認するために、ログ出力を記述します。ログ出力は、`Logger.log()` で行えます。いわゆる JavaScript における

`console.log()` のようなものです。`Logger.log()` は、文字列を渡せば文字列が表示されますし、変数を渡せばその中身が表示されます。

ひとまず実行できていることが確認できればよいので、以下のようにログ出力を記述します。中に入る文字列は当然なんでもよいです。

```
1 function slackSender() {  
2   Logger.log("My first logging!!");  
3 }
```

ソースコード 7 Logging

上記のようにスクリプトが書けたら、実行してみましょう。実行は、エディタ画面上部にあるツールバーの、再生ボタン (▶ ボタン) をクリックします。複数の `function` が定義されている場合は、実行する `function` を右側のドロップダウンから選択できます。

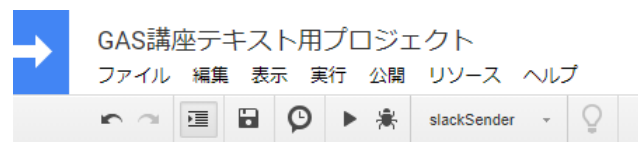


図 14 GAS ツールバー

初めての実行時、**Authorization required** といったタイトルのダイアログが表示されます。これは、「このスクリプトを実行していいですか?」という確認と、それをユーザーの G Suite アカウントから承認するダイアログです。画面の指示に従って承認してください。詳しくは、[こちらのウェブサイト](#)が詳しいので、必要に応じて参照してください。

実行が完了したら、ログを確認してみましょう。ログは、ツールバーの**表示**から**ログ**をクリックするか、**Ctrl+Enter** で表示できます。正常に実行できていれば、以下の図 15 のようにログが出力されています。`Logger.log()` を使ったログ出力はスクリプトのデバッグによく使用するので、覚えておきましょう。



図 15 ログ出力例

3.1.2 UriFetchApp と Web API への HTTP リクエスト

ここから本格的に Google Apps Script 上で Incoming Webhooks を利用していきます。これまで、Incoming Webhooks を「Webhook URL に投稿したい内容を持った HTTP リクエストを送ると、Slack にメッセージが投稿できる」と説明していましたが、ここからは「投稿したい内容を持った HTTP リクエスト」を送る具体的な方法について説明していきます。

Google Apps Script では、**UriFetchApp** というクラスで HTTP リクエストを行います。このクラスでは、ウェブを介した HTTP リクエストと、そのレスポンスの取得が行えます。HTTP リクエストを送るには、**UriFetchApp.fetch(url, params)** を使用します。今回、リクエストを送る先の URL は Webhook URL に対応しますので、**url** 引数にはこれを入力します。**param** 引数については後述します。

さて、リクエストを受け取る側（Slack のサーバー）もプログラムで動いているので、闇雲に HTTP リクエストを送ってもサーバー側は解釈できません。リクエストの形式には決まり（プロトコル）が定められています。2019 年現在、Web サービスの提供する Web API の多くは JSON を受け取り、JSON を返すように作られています。Incoming Webhooks でも、送信したい内容の JSON を送ることで、Slack のサーバー側が正しく解釈でき、メッセージを投稿できます。

Incoming Webhooks に渡す JSON のうち、最も単純なものは以下に示すような「投稿したいメッセージ」のみを持った JSON です。これを Webhook URL に対して送ると、デフォルトの Slack チャンネルにメッセージが送信されます。

```
1 {  
2   "text": "This is the message"  
3 }
```

ソースコード 8 Minimum JSON payload for Incoming Webhooks

Slack のサーバー側へ命令を送る際の決まりは、JSON で送ることでしたが、その「送る」を担う **HTTP リクエスト** にも決まりが存在します。この「決まり」に即してパラメータを設定する部分が、冒頭で紹介した `UrlFetchApp.fetch(url, params)` の `params` 引数に対応します。ここで必要になるのは、「JSON 形式でこの (上記の JSON) データを送りますよ」ということの宣言です。この宣言は、以下のように分割することができ、またそれぞれが `params` 引数のパラメータに対応します。

- JSON 形式で
 - `headers` パラメータに対応
- このデータを
 - `payload` パラメータに対応
- 送りますよ
 - `method` パラメータに対応

まず、「送ります」は `POST` というリクエストメソッドに対応します。ここでは `method` に `POST` を指定すればいいんだな程度で大丈夫です。リクエストメソッドについて気になる方は各自調べていただければと思います。HTTP リクエストメソッドには他に `GET`, `PUT`, `DELETE` などがあり、Web API ではこれらのメソッドによってサービスに対する操作を区別することが多いです。

「JSON 形式」を宣言するには、リクエストヘッダに `Content-type` を指定します。JSON 形式に対応する `Content-type` は `application/json` です。リクエストヘッダについて気になる方は (以下略)

したがって、以下のような形式になります。

```
1 {  
2   "method": "POST",  
3   "headers": {"Content-type": "application/json"}  
4 }
```

ソースコード 9 params argument for `UrlFetchApp`

さて、あとはソースコード 8 に示した JSON オブジェクト (実際に API に対して渡すデータ) を `params` 引数に含めれば、`params` に渡すべきオブジェクトは完成します。これは、`payload` パラメータに設定するのですが、`payload` パラメータに設定できるのは文字列のみです。そこで、先の JSON オブジェクト全体を文字列に変換する必要があります。これは、JavaScript 組み込みの `JSON.stringify` という関数で変換できます。ある JSON オブジェクト `obj` を JSON 文字列に変換するには、この関数に `obj` を渡してやればよいので、`JSON.stringify(obj)` となります。したがって、スクリプト全体は以下ようになります。
`your/webhook/url` の部分は実際に取得した Webhook URL を入力します。

```
1 function slackSender() {  
2  
3   var payload = {  
4     "text": "This is the message"  
5   }  
6  
7   var options = {  
8     "method": "POST",  
9     "headers": {"Content-type": "application/json"},  
10    "payload": JSON.stringify(payload)  
11  };  
12  
13  UrlFetchApp.fetch("your/webhook/url", options);  
14 }
```

上記スクリプトを実行すると、以下のように Slack に投稿されます。

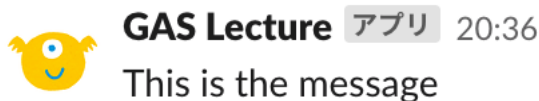


図 16 スクリプト実行例

3.2 いろいろな Payload の作成と実行例

前節では、`text` に単純な文字列を指定して、シンプルなメッセージ投稿を行いました。しかし、Incoming Webhooks では、通常私たちが Slack の画面から投稿するよりもリッチな投稿ができます。この節ではその例について紹介します。

3.2.1 mrkdwn 記法

通常私たちが Slack のメッセージを投稿するときも使用できるように、Slack には Markdown に似た記法でテキストの装飾ができます。この記法を **mrkdwn** と呼びます。

たとえば、図 17 に示すように、`*bold*` と文字列を `*` で囲むと、囲まれた文字列が太字になったり、`'code'` と文字列を `'` (バッククオート) で囲むと、囲まれた文字列がコードテキストになったりするものです。

```
*bold*  
_italic_  
~cancel~  
`code`
```

と書くと

bold

italic

~~cancel~~

`code`

と表示されます。

図 17 mrkdwn 記法の例

これは Incoming Webhooks でメッセージを投稿する場合も有効で、実際以下のソースコード 13 のような payload でメッセージを投稿すると、図 18 のように表示されます。

```

1 {
2   "text": "**This* is a _sample_ of 'mrkdwn' ~notation~"
3 }

```

ソースコード 11 mrkdwn payload sample

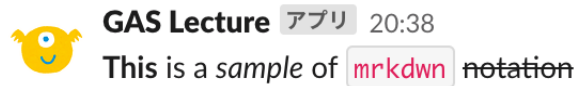


図 18 mrkdwn 記法を使ったメッセージ投稿例

3.2.2 メンション

前項では、Slack の画面から通常通りメッセージを投稿するように、Incoming Webhooks でもメッセージの装飾ができることを紹介しました。一方、@here、@channel、@everyone を含む、メンションには少し注意が必要です。

たとえば、私 (@hitsumabushi845) にメンションを飛ばそうと思い、以下のような payload を作成したとします。

```

1 {
2   "text": "@hitsumabushi845 mention...?"
3 }

```

ソースコード 12 mention payload sample1

この payload を使って Incoming Webhooks でメッセージを投稿すると、以下のように正しくメンションにならず、@hitsumabbushi845 は単なる文字列として表示されてしまいます。

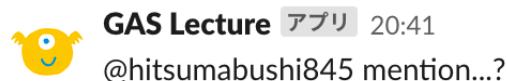


図 19 Incoming Webhooks を使ったメンション（失敗例）

これを正しくメンションとして表示するためには、以下のように@username を<>で囲む必要があります。なお、@username を<>で囲むメンションは～

```

1 {
2   "text": "<@hitsumabushi845> mention!"
3 }

```

ソースコード 13 mention payload sample2

こうすることで、正しくメンションとしてメッセージが投稿されます。

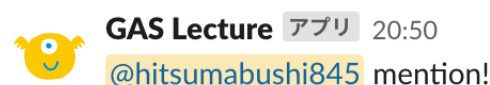


図 20 Incoming Webhooks を使ったメンション（成功例）

同様に, @here, @channel, @everyone も<@here>, <@channel>, <@everyone>となる・・・かと思いきや, これらは<!here>, <!channel>, <!everyone>です。

表 1 Incoming Webhooks でのメンション対応表

対象	通常	Incoming Webhooks
個別のユーザー	@username	<@username>
チャンネルに参加しているオンライン状態のユーザー全員	@here	<!here>
チャンネルに参加しているユーザー全員	@channel	<!channel>
ワークスペースに参加しているユーザー全員 ^{*4}	@everyone	<!everyone>

なお, username を使ったメンションは Slack API 経由では**使用できません** (Incoming Webhooks では可能)。

3.2.3 リンク

ここからは, 通常私たちが Slack でメッセージを投稿する際には使えないが, Incoming Webhooks なら使える機能を紹介します。まずはリンクです。Incoming Webhooks 経由のメッセージでは, 文字列にリンクを埋め込むことができます。実際に Incoming Webhooks で文字列にリンクを埋め込む例を見てみましょう。

以下のように, <{URL}|{リンクを埋め込みたい文字列}>を投稿する payload に渡すと, Slack 上では図 21 のように文字列にリンクが埋め込まれて表示されます。

```
1 {  
2   "text": "<https://www.google.co.jp/|Linked text>"  
3 }
```

ソースコード 14 Link payload sample



図 21 Incoming Webhooks を使ったリンクの埋め込み

文字列にリンクを埋め込む必要がなく, URL をそのまま投稿する形で良ければ, 特に<>で囲む必要もなく, URL をそのまま text の文字列に含めてしまえば大丈夫です。

3.2.4 Block Kit

2019 年 2 月に登場した **Block Kit** を紹介します。Block Kit は, アプリケーションから Slack へ投稿するリッチなメッセージを組み立てる UI フレームワークです。フレームワークといっても, API を介してメッセージを投稿する際の形式の一つですので, ライブラリのインストールなどといった環境構築は必要ありません。これまでと同様に payload に含めるだけで利用できます。

Block Kit を使うことで, 図 22 のような, 複雑かつリッチなメッセージを組み立てることができます。ボタンや画像, 区切り線に至るまで, すべて Block Kit を使って構築されています。

^{*4} @everyone はワークスペース全体が参加するチャンネル (#general, #random など) でのみ使うことができます。

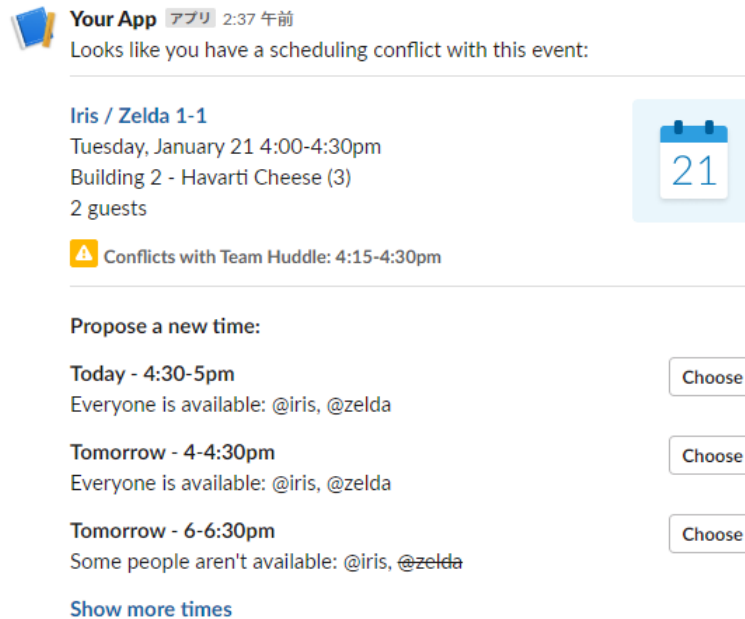


図 22 Block Kit でのメッセージ作成例

Block Kit の要素は大きく分けて以下の 5 つです。

- **Section**
主にテキストの表示に使用されるが、画像やボタンを含めることもでき、最も自由に扱える要素。
- **Context**
Section として表示されるよりもサイズが小さく、色が薄いテキスト要素。主に説明などの記載に使用される。
- **Image**
画像要素。キャプションもつけられる。
- **Divider**
分割線。
- **Actions**
ボタンやセレクトボックス、日付セレクトなどのアクション要素。主に Slack とアプリケーション間の双方向通信が可能な場合に使用される。Incoming Webhooks はアプリケーションから Slack への単方向通信なので、このテキストでは扱わない。

Block Kit は、これまでの `text` のみの `payload` ではなく、要素ごとにオブジェクトを作成し、それを格納したリストで表現されます。そのため、目的の見た目を実現するために JSON を手入力で作成するのは骨が折れる作業です。Slack は Block Kit のプロトタイピングツール **Block Kit Builder** を提供しています。これは、GUI を用いてメッセージのひな形を作成できるツールです。

左カラムから追加したい要素を選ぶと、中央カラムに Slack に投稿された際のイメージが表示され、実際の JSON は右カラムに表示されます。右カラムの JSON は編集可能なので、デフォルトの文字列から変更したい場合はこちらを編集して変更します。そのため、Block Kit を利用する場合は Block Kit Builder を使ってひな

形を作り、ひな形をもとにプログラムに落とし込む、といったフローが最も簡単だと思います。

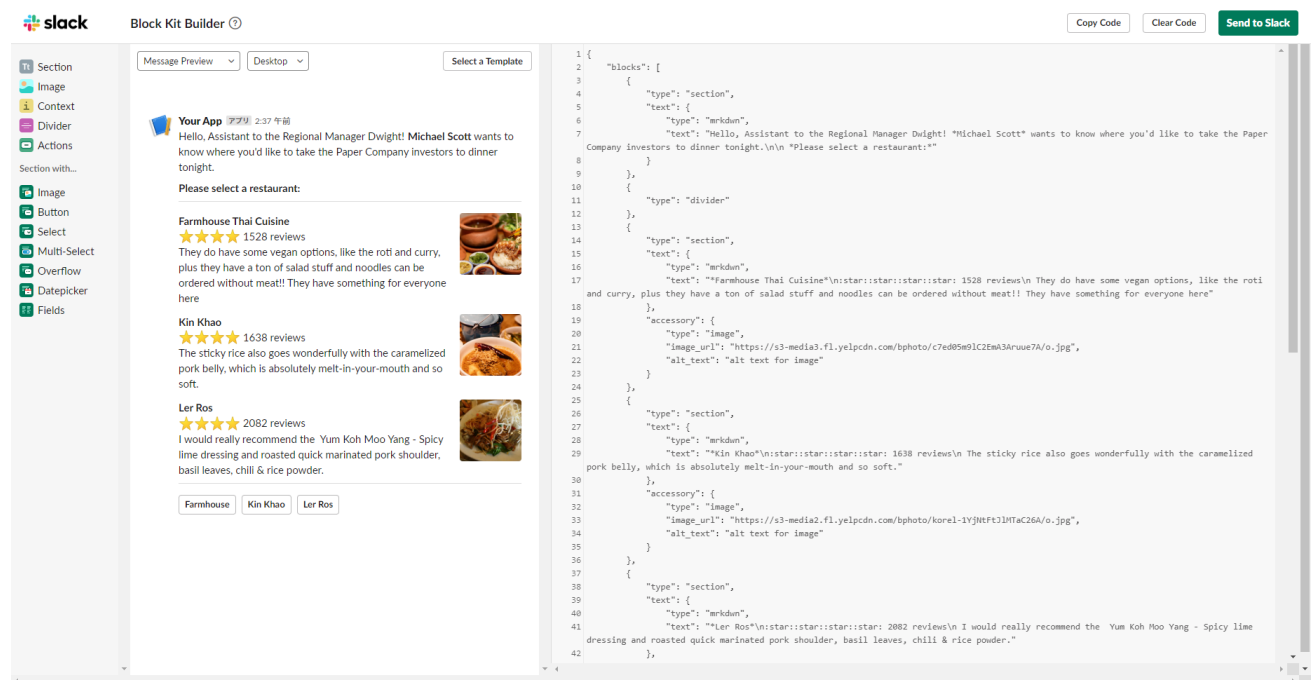


図 23 Block Kit Builder 画面

3.2.5 Attachments

Block Kit が登場するまで、この Attachments がリッチなメッセージを作成する手段でした。しかし、Attachments は Block Kit の登場によって **Legacy** とされ、outmoded approach（廃れた手法）とまで記載されるようになってしまいました。

しかし、Attachments にできて Block Kit にはできないことが 1 つだけあります。それは、“メッセージの左側に色付きのバーを表示すること”です。Attachments には、Block Kit Builder のような GUI プロトタイピングツールは提供されていませんが、現在の Attachments は Block Kit のラッパーとしても提供されており、Block Kit で作成した blocks 要素を attachments の各要素に入れ込むだけで Attachments が利用できます。つまり、Attachments は Block Kit で作成されたメッセージの左側に色付きのバーを追加する機能、として捉えるのがよさそうです。

Attachments は、オブジェクトのリストで提供されます。したがって、1 つの Attachment を投稿する場合は、以下のような payload になります。

```
1 {
2   "text": "Main text",
3   "attachments": [
4     {
5       "color": "#2FA44F",
6       "blocks": [
7         {
8           "type": "section",
```

```

9      "text": {
10         "type": "mrkdwn",
11         "text": "This is a block component"
12     }
13 }
14 ]
15 }
16 ]
17 }

```

ソースコード 15 Attachments payload sample

`attachments` 要素の内容のほとんどが Block Kit と同様であることがわかります。違いは `color` 要素の有無です。この payload を使って、Slack にメッセージを投稿すると、以下のように表示されます。

Main text

■ This is a block component

図 24 Attachments を使った投稿例

4 Google Forms と GAS

これまでは Google Apps Script と Incoming Webhooks だけを使って、Slack にいろいろなメッセージを投稿してきました。ここからは、実際に Google Apps との連携プログラムを書きます。まずは、Google Forms を題材に GAS を作成していきましょう。

この章では、「フォームが送信されたら Slack にその内容を通知する」スクリプトを作成します。Google Forms では、Google Spreadsheets に回答内容を蓄積する機能はありますが、Slack への通知機能はありません。問い合わせフォームや依頼フォームなど、回答が送信されたら即座にキャッチしたいケースは多くあるかと思います。ぜひ、GAS を使って Slack 通知を実装してみてください。

■本章で扱うトピック

- Google Apps に紐づくスクリプトとイベントトリガ
- Google Forms における Event Object

なお、本テキストでは Google Forms そのものの利用方法は紹介しません。そのため、断りなしに作成済みのフォームを使用する場合があります。

4.1 FormApp

GAS における Google Forms への API は **FormApp** というクラスで提供されます。**FormApp** は、スクリプトによるフォームの作成、編集、そして参照の手段を提供します。つまり、フォームそのものに対する操作を提供する API となります。実際にフォームから送信された回答を取得する手段は **FormApp** は持っていません。これについては次節 4.2 以降で扱います。

普段私たちが Google Forms でフォームを作成するときは Web 画面からインタラクティブに作成すること

と思います。わざわざスクリプトを書いてフォームを作成・編集するメリットはイメージしづらいかもしれませんが。例えば、セレクトボックスの要素を何らかのスプレッドシートやデータベースに記録されている値から動的に生成したい場合などにおいて、スクリプトを用いたフォーム作成が効果を発揮します。ここでも、“Google Forms とスプレッドシート” といった、「異なるサービス間の連携」において Google Apps Script が効果を発揮することがわかるかと思います。

4.1.1 FormApp を使ったフォーム作成

この項では、テキスト入力形式の質問とセレクトボックス形式の質問を持ったフォームを、Google Apps Script で作成してみます。これまでと同様に、Google Apps Script のプロジェクトを作成します。今回は、関数名を `createForm` とします。

```
1 function createForm() {  
2  
3 }
```

ソースコード 16 FormApp sample 1

画面からフォームを作成する場合も、Google Apps Script でフォームを作成する場合も、基本的な流れは変わりません。**新しいフォームを作成**して、作成したフォームに**質問を追加**するといった流れです。

まず、新しいフォームを作成する場合は、`FormApp.create()` を使用します。引数には、作成するフォームの名称を入力します。

```
1 function createForm() {  
2   var form = FormApp.create("Sample Form");  
3 }
```

ソースコード 17 FormApp sample 2

上記の状態でスクリプトを実行すると、`Sample Form` という名称で空のフォームが作成されます。なお、このスクリプトは**実行するごとに新しいフォームが作成されますので、注意が必要です**。

ここから、フォームに対して質問を追加していきます。まずは、テキスト入力形式の質問を追加します。テキスト入力形式の質問には、短文回答（1 行）と長文回答（複数行）の 2 種類があります。今回は、長文回答形式の質問を追加します。

長文回答形式の質問を追加するには、フォームに対して、`addParagraphTextItem()` を実行します。

```
1 function createForm() {  
2   var form = FormApp.create("Sample Form");  
3   form.addParagraphTextItem()  
4 }
```

ソースコード 18 FormApp sample 3

ここで追加された質問には、タイトルや説明、必須項目か否かなどの情報はまだ設定されていません。`Form.addParagraphTextItem()` の返り値は `ParagraphTextItem` クラスです。このクラスには、以下のようなメソッドがありますので、これを使って設定していきます。

- `setTitle()`
タイトルを設定する
- `setHelpText()`
説明を設定する

- `setRequired()`
必須項目として設定する

```

1 function createForm() {
2   var form = FormApp.create("Sample Form");
3   form.addParagraphTextItem()
4     .setTitle("Paragraph Text Question")
5     .setHelpText("This ParagraphTextItem is automatically created by GAS.")
6     .setRequired(true);
7 }

```

ソースコード 19 FormApp sample 4

上記スクリプトを実行すると、以下の図 25 のようなフォームが作成されます。

図 25 FormApp によるフォームの作成例 1

さらに、セレクトボックス形式の質問を追加していきます。セレクトボックス回答形式の質問を追加するには、フォームに対して、`addListItem()` を実行します。

タイトルや説明の設定は、`ParagraphTextItem` と同様に `setTitle()`、`setHelpText()` で行えます。セレクトボックスにアイテムを追加するのは `ListItem.setChoiceValues()` が最も簡便な手法です。これは、文字列のリストをまとめてアイテムとして追加するメソッドです。

```

1 function createForm() {
2   var form = FormApp.create("Sample Form");
3
4   // 長文回答形式の質問を追加する
5   form.addParagraphTextItem()
6     .setTitle("Paragraph Text Question")
7     .setHelpText("This ParagraphTextItem is automatically created by GAS.")

```

```

8      .setRequired(true);
9
10     // セレクトボックス形式の質問を追加する
11     form.addListItem()
12     .setTitle("List Question")
13     .setHelpText("This ListItem is automatically created by GAS.")
14     .setChoiceValues(["hoge", "fuga", "foo", "bar"]);
15 }

```

ソースコード 20 FormApp sample 5

以上でフォームの作成、質問の追加が完了しました。最終的に作成されたフォームは以下の図 26 です。他の質問形式の追加方法など、FormApp のより詳しい情報は、[公式ドキュメント](#)に記載されています。

図 26 FormApp によるフォームの作成例 2

最後に、作成したフォームの URL を取得する方法を紹介します。作成したフォームの URL は、`form.getPublishedUrl()` と `form.getEditUrl()` から取得できます。それぞれ、実際に公開されるフォームの URL と、フォームの編集画面の URL です。

```

1 function createForm() {
2     var form = FormApp.create("Sample Form");
3
4     // 長文回答形式の質問を追加する
5     form.addParagraphTextItem()
6     .setTitle("Paragraph Text Question")
7     .setHelpText("This ParagraphTextItem is automatically created by GAS.")
8     .setRequired(true);

```

```

9
10 // セレクトボックス形式の質問を追加する
11 form.addItem()
12 .setTitle("List Question")
13 .setHelpText("This ListItem is automatically created by GAS.")
14 .setChoiceValues(["hoge", "fuga", "foo", "bar"]);
15
16 Logger.log(form.getPublishedUrl());
17 Logger.log(form.getEditUrl());
18 }

```

ソースコード 21 FormApp sample 6

上記のスクリプトを実行すると、ログに作成されたフォームの URL が表示されます。

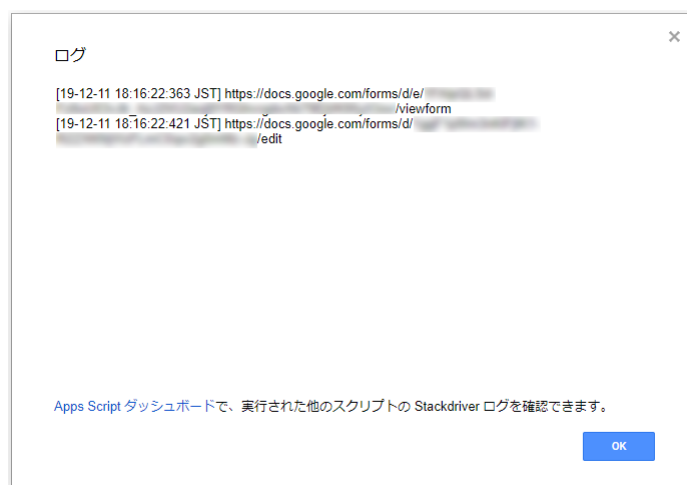


図 27 フォーム URL の取得例

4.2 Google Apps に紐づくスクリプト

前節では、FormApp を使ってスクリプトからフォームを作成する方法を紹介しました。本節では、既に作成されたフォームから回答が送信されたとき、それを受け取る方法について紹介します。今回は、前節で作成したフォーム (図 26) を引き続き使用します。

まず、フォーム編集画面右上にある、「⋮」をクリックし、「スクリプト エディタ」を選択します。すると、スクリプト編集画面が開きます。これは、フォームに紐付いたスクリプトになりますので、以降はこちらからスクリプトを編集します。

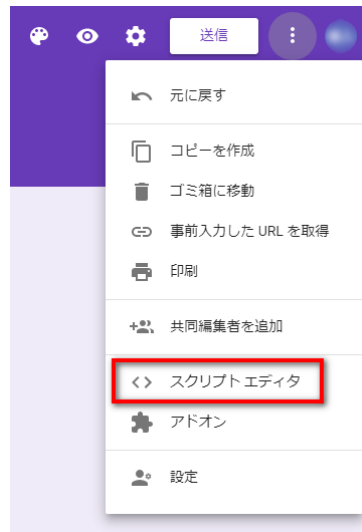


図 28 フォームからスクリプトエディタを開く

4.2.1 イベントトリガ

フォームをはじめとして、Google Apps に紐づくスクリプトでは、**イベントトリガ**が使用できます。イベントトリガとは、Google Apps に対するユーザーの操作^{イベントトリガ}をきっかけにスクリプトを実行する機能です。フォームであれば、「回答の送信」や「フォームを開く」、スプレッドシートであれば「シートの編集」や「シートの追加」がイベントにあたります。

イベントトリガや、後述する時間主導型トリガなどのトリガ編集画面は、スクリプト編集画面のツールバーから、時計マークのボタンをクリックするとアクセスできます。トリガ編集画面右下に「+ トリガーを追加」というボタンがあります。これをクリックして、トリガを設定します。



図 29 トリガ編集画面を開く

トリガ設定画面では、以下の項目を設定できます。

- 実行する関数
トリガによって実行される関数を選択します。
- 実行するデプロイ
スクリプトのバージョン管理を行っている場合、ここから実行するバージョンを選択できます。
常に最新のスクリプトを実行するのであれば、ここは **Head** のままで大丈夫です。
- イベントのソース
イベントをどこから取得するか選択します。今回は、フォームに回答が送信されたらスクリプトを実行しますので、「フォームから」を選択します。

- Google Apps から（フォームから）
- 時間主導型
- カレンダーから
- イベントの種類
「イベントのソース」にて「Google Apps から」を選択した場合、こちらが表示されます。フォームの場合、「起動時」と「フォーム送信時」が選択できます。今回は後者を選択します。
- エラー通知の頻度
トリガによるスクリプトの実行でエラーが発生した場合に、メール通知を行うことができます。以下の頻度から選択できます。
 - 今すぐ通知を受け取る
 - 1 時間おきに通知を受け取る
 - 毎日通知を受け取る
 - 1 週間おきに通知を受け取る

今回は、以下のように設定を行いました。

GAS講座テキスト用プロジェクト（Form）のトリガ...

実行する関数を選択

getAnswer ▼

実行するデプロイを選択

Head ▼

イベントのソースを選択

フォームから ▼

イベントの種類を選択

フォーム送信時 ▼

エラー通知設定 +

毎日通知を受け取る ▼

キャンセル 保存

図 30 トリガの設定例

4.2.2 Event Object

イベントトリガによって実行された関数は、そのイベントに関する情報を関数の引数として受け取ることができます。これを **Event Object** といいます。Event Object を受け取るために、実行される関数に引数を設定します。いま、「フォーム送信時」のイベントをトリガとして設定しているので、それに対応する引数として、`formSendEventObject` と名前をつけます。

```
1 function getAnswer(formSendEventObject) {  
2  
3 }
```

ソースコード 22 Event Object sample 1

Event Object は、対応するイベントに応じて保持している情報が異なります。フォーム送信イベントによって受け取る Event Object にはフォームの回答、スプレッドシートの編集イベントによって受け取る Event Object には編集された内容・・・といった具合です。今回はフォーム送信による Event Object を受け取っていますので、[こちら](#)に記載されている通りの内容が含まれています。実際に送信されたフォームの内容は、`response` 要素に含まれています。`response` 要素には、フォームの回答を格納するための **FormResponse** クラスのオブジェクトが含まれています。**FormResponse** クラスのオブジェクトには、実際に送信された回答の内容、送信された時間、送信した人のメールアドレス*5などが格納されています。まずは、送信された回答を取得してみましょう。

■**回答の内容を取得する** 実際の回答を取得するには、`FormResponse.getItemResponses()` を実行します。これは、個々の回答を格納するための **ItemResponse** クラスのリストが返ってきます。

```
1 function getAnswer(formSendEventObject) {  
2     var response = formSendEventObject.response;  
3     var itemResponses = response.getItemResponses();  
4 }
```

ソースコード 23 Event Object sample 2

`itemResponses` 変数を `Logger.log(itemResponse)` でログに出力してみましょう。イベントトリガによって実行されるスクリプトのログは、エディタ画面を開いた状態でイベントを起こすと確認できます。

図 26 のフォームを使用していますので、質問の数は 2 つです。`itemResponses` をそのままログに出力すると、以下のように出力されます。

```
1 [19-12-12 16:34:33:783 JST] [ItemResponse, ItemResponse]
```

ソースコード 24 Event Object output example 1

2 つの回答が含まれているので、要素を 2 つ持ったリストになります。しかし、このままでは **ItemResponse** クラスのオブジェクトが 2 つあることしか分からず、どのような回答が得られたのかわかりません。

ItemResponse オブジェクトから、実際の回答内容を取得するには、`ItemResponse.getResponse()` を実行します。いま、**ItemResponse** オブジェクトのリストとして回答を取得できていますので、以下のようなスクリプトで順にログに出力してみます。

*5 メールアドレスを収集する設定が有効になっている場合のみ

```

1 function getAnswer(formSendEventObject) {
2   var response = formSendEventObject.response;
3   var itemResponses = response.getItemResponses();
4
5   for (var i = 0; i < itemResponses.length; i++) {
6     Logger.log(itemResponses[i].getResponse());
7   }
8 }

```

ソースコード 25 Event Object sample 3

上記のスクリプトを保存した状態でフォームから回答を送信すると、以下のようなログが出力されます。しかし、今取得できている（ログに表示できている）のは回答の内容のみであり、「どのような質問への回答なのか」が分かりづらく、不親切なログとなっています。

```

1 [19-12-12 16:40:10:881 JST] Test Answer
2 [19-12-12 16:40:10:881 JST] hoge

```

ソースコード 26 Event Object output example 2

そこで、ログに質問のタイトルも出力するようにしてみましょう。
質問のタイトルは、`ItemResponse.getItem().getTitle()` で取得できます。これは、まず `ItemResponse.getItem()` で質問の内容を持つ `Item` オブジェクトを取得し、それからタイトルを取得しています。ログを読みやすくするため、質問と回答を区別できるような文言を追加し、以下のようなスクリプトを書きます。

```

1 function getAnswer(formSendEventObject) {
2   var response = formSendEventObject.response;
3   var itemResponses = response.getItemResponses();
4
5   for (var i = 0; i < itemResponses.length; i++) {
6     Logger.log("Question: " + itemResponses[i].getItem().getTitle());
7     Logger.log("Answer: " + itemResponses[i].getResponse());
8   }
9 }

```

ソースコード 27 Event Object sample 4

上記スクリプトが実行されると、以下のようなログになります。だいぶ読みやすくなりました。

```

1 [19-12-12 00:49:36:146 PST] Question: Paragraph Text Question
2 [19-12-12 00:49:36:147 PST] Answer: Test Answer
3 [19-12-12 00:49:36:312 PST] Question: List Question
4 [19-12-12 00:49:36:314 PST] Answer: hoge

```

ソースコード 28 Event Object output example 3

■回答者のメールアドレスを取得する 次に、回答した人のメールアドレスを取得してみましょう。メールアドレスを取得するには、フォームの設定から「メールアドレスを収集する」にチェックを入れておく必要があります。

回答者のメールアドレスは、`FormResponse.getRespondentEmail()` から取得できます。Works Human Intelligence では、メールアドレスの@前が Slack の Username にもなっていますので、回答者にメンションを送るなどといった応用の足がかりになります。

先程のスクリプトに、メールアドレスのログ出力を追加します。

```

1 function getAnswer(formSendEventObject) {
2     var response = formSendEventObject.response;
3     var itemResponses = response.getItemResponses();
4
5     Logger.log("RespondentEmail: " + response.getRespondentEmail());
6
7     for (var i = 0; i < itemResponses.length; i++) {
8         Logger.log("Question: " + itemResponses[i].getItem().getTitle());
9         Logger.log("Answer: " + itemResponses[i].getResponse());
10    }
11 }

```

ソースコード 29 Event Object sample 5

上記スクリプトを実行すると、以下のようなログが出力されます。メールアドレスが正しく取得できることが確認できます。

```

1 [19-12-12 17:56:50:299 JST] RespondentEmail: shimada_ken@works-hi.co.jp
2 [19-12-12 17:56:50:441 JST] Question: Paragraph Text Question
3 [19-12-12 17:56:50:442 JST] Answer: Test Message
4 [19-12-12 17:56:50:535 JST] Question: List Question
5 [19-12-12 17:56:50:536 JST] Answer: hoge

```

ソースコード 30 Event Object output example 4

FormResponse クラスには、他にもいろいろな情報が格納されています。詳細は [こちら](#) から確認できます。

4.3 質問形式と取り出され方

前節では、文字列形式の回答と択一方式の回答についてログに出力してみました。フォームには他にも複数選択可能な質問（チェックボックスなど）や、日時を選択する質問など、様々な形式の質問があります。これらの回答をスクリプトで取り出すとどうなるか確認してみましょう。

図 31 のような、様々な形式の質問をもったフォームを作成してみました。これの回答を先ほどと同様にログに出力してみます。

Sample Form

ファイルをアップロードしてこのフォームを送信すると、Google アカウントに関連付けられている名前、ユーザー名、写真が記録されます。shimada_ken@works-hi.co.jp が正しいアカウントでない場合は、[アカウントを切り替えて](#)ください

チェックボックス形式の質問

☐ 選択肢 1

☐ 選択肢 2

☐ 選択肢 3

☐ その他: _____

日時の質問

日付 時刻

年 / 月 / 日 : _____

グリッドチェックボックス

	1 列目	2 列目
1 行目	<input type="checkbox"/>	<input type="checkbox"/>
2 行目	<input type="checkbox"/>	<input type="checkbox"/>

グリッドラジオボタン

	1 列目	2 列目
1 行目	<input type="radio"/>	<input type="radio"/>
2 行目	<input type="radio"/>	<input type="radio"/>

ファイルのアップロード

[ファイルを追加](#)

Google フォームでパスワードを送信しないでください。

図 31 いろいろな形式の質問を持ったフォーム

上記のようなフォームの回答をスクリプトから取り出すと、以下のような結果が取得できます。

```

1 [19-12-12 01:40:55:372 PST] RespondentEmail: shimada_ken@works-hi.co.jp
2 [19-12-12 01:40:55:446 PST] Question: チェックボックス形式の質問
3 [19-12-12 01:40:55:447 PST] Answer: 選択肢 2, その他
4 [19-12-12 01:40:55:520 PST] Question: 日時の質問
5 [19-12-12 01:40:55:521 PST] Answer: 2019-12-12 12:34
6 [19-12-12 01:40:55:610 PST] Question: グリッドチェックボックス
7 [19-12-12 01:40:55:610 PST] Answer: 1 列目, 2 列目, 2 列目
8 [19-12-12 01:40:55:676 PST] Question: グリッドラジオボタン
9 [19-12-12 01:40:55:677 PST] Answer: 1 列目, 2 列目

```

10 [19-12-12 01:40:55:744 PST] Question: ファイルのアップロード
11 [19-12-12 01:40:55:745 PST] Answer: 1P0g8_AFoWIob4uFeHxQ9ZnUK3eo_Jqyw

ソースコード 31 Event Object sample 6

質問の回答は、`String`、`String[]`、または `String[][]` のいずれかの形式で取得されます。基本的には `String` で取得されますが、複数の回答を持ちうる以下の 3 つは `String[]` または `String[][]` で取得されます。詳細は [こちら](#)。

- `CheckboxItem` (チェックボックス形式の質問)
この回答は `String[]` (文字列のリスト) で返ってきます。
- `GridItem` (グリッドラジオボタン)
この回答も `String[]` で返ってきます。[1 行目の回答, 2 行目の回答, ...] といった具合です。
- `CheckboxGridItem` (グリッドチェックボックス)
この回答は、各行で複数の回答が選択されうるので、唯一 `String[][]` (文字列の 2 次元リスト) で返ってきます。
 i 行目の j 番目の回答を $A_{i,j}$ とすると、 $[[A_{1,1}, A_{1,2}, \dots], [A_{2,1}, A_{2,2}, \dots], \dots]$ といった形です。

また、「ファイルのアップロード」の質問の取り出され方には少し注意が必要です。上記ログの最終行を見るとわかるように、謎の文字列が返ってきているように見えます。これは **ファイル ID** と呼ばれる値で、Google Drive にアップロードされるファイルに一意に与えられる文字列です。Google Drive で、なんらかのファイルの共有リンクを取得してみてください。<https://drive.google.com/open?id={ファイル ID}> という URL が取得できると思います。この “id=” 以下の文字列がファイル ID です。Google Drive にアップロードされたファイルの URL はファイル ID で識別されていますので、逆にファイル ID さえあれば、実際にブラウザからファイルにアクセスするための URL を特定できます。

ちなみに、ファイル ID のみならず、スプレッドシートやスライド、フォームなど、Google Apps で作成したものにはすべて ID が割り当てられています。

4.4 Practice: Google Forms × Slack

本章では、Google Forms を GAS から扱う方法について説明してきました。また、前章では GAS から Slack へメッセージを投稿する方法について説明しました。これらを組み合わせて、**図 32 に示すようなフォームを作成し、回答が送信されたら、その内容を Slack に送信する**スクリプトを作成してみましょう。

作成例は本テキストの付録に記載しています。これが唯一の正解というわけではないので、参考程度にとどめておいてください。

GAS×Slack Practice Form

このフォームを送信すると、メールアドレス（shimada_ken@works-hi.co.jp）が記録されます。
自分のアカウントでない場合は、[アカウントを切り替えて](#)ください

***必須**

今日のごはんは *

☐ 肉

☐ 魚

☐ 鍋

☐ ラーメン

☐ 食べない

☐ その他: _____

☐ 回答のコピーを自分宛に送信する

送信

Google フォームでパスワードを送信しないでください。

図 32 対象となるフォーム

5 Google Spreadsheets と GAS

本章では、Google Spreadsheets を操作する GAS について紹介します。

■本章で扱うトピック

- Spreadsheet の構造
- SpreadsheetApp
- 時間主導型トリガ

5.1 Spreadsheet の構造 - プログラムでスプレッドシートを扱うとはどういうことか

普段私たちがスプレッドシートを扱うとき、その構造について意識することはありません。といっても、実際にはスプレッドシートというアプリケーションそのものはその構造の上に成り立っています。なぜ私たちがこの構造を意識せず使用できるかというと、構造を視覚を用いて無意識的に認識しているからにほかなりません。

しかし、プログラムは視覚を持ちません。ですので、これからプログラムを作成する私たちはスプレッドシートの構造について改めて認識する必要があります。

さて、スプレッドシートは何から構成されているのでしょうか？大まかに分けると、以下のような要素から構成されています。

- スプレッドシート全体
- シート
- 行や列
- セル
- 値

これらは、上から下に向かって包含関係を持ちます。つまり、「スプレッドシート全体」には1つ以上の「シート」が含まれ、「シート」には1つ以上の「行や列」が含まれ・・・といった関係です。スプレッドシート上のある「値」をプログラムを使って取り出す場合は、この包含関係を意識する必要があります。

つまり、いきなり「値」を取り出すことはできないので、「スプレッドシート全体」を取ってきて、そこから「シート」を取ってきて、そこから「行や列、セル」を取ってきて、ようやく「セル」の中にある「値」にアクセスできます。マトリョーシカのように順番に取り出すしかないのです。マトリョーシカのように^{*6}。

^{*6} 大事なことなので2回言いました



図 33 プログラムでスプレッドシートから値を取り出す順序（イメージ）

ところで、先程“私たちは（スプレッドシートの）構造を視覚を用いて無意識的に認識している”と説明しました。この具体的な例について紹介しましょう。

あなたがあるスプレッドシートに記録されている情報を確認したいとき、以下のような行動を取ると思います。

1. まず、対象のスプレッドシートを開こうと、ブックマークや Google Spreadsheets のトップページを参照します。
2. 対象のスプレッドシートを開いたら、（シートが複数ある場合）シート名から対象のシートを選択します。
3. 対象のシートを開いたら、視覚を用いて目的の情報を探します。

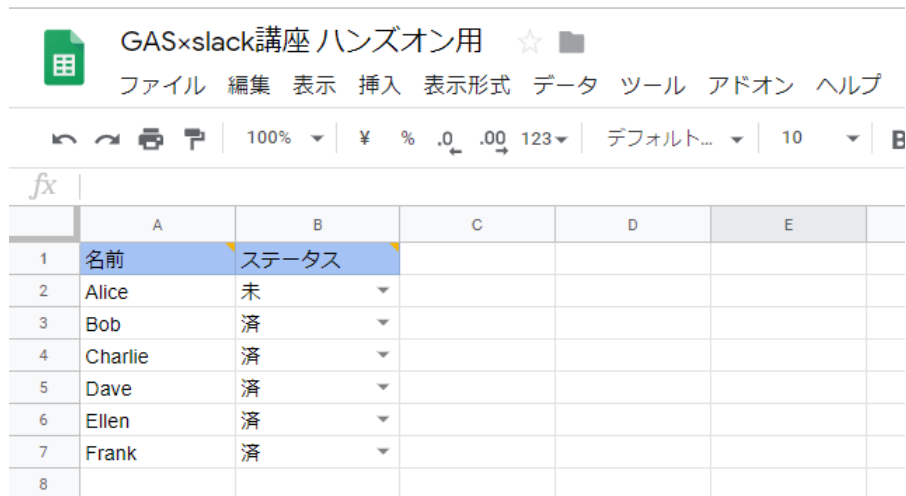
上記のフローはプログラムが行っている方法とほぼ同じと言えます。しかし、この構造について意識することは普段ありません。

5.2 SpreadsheetApp

フォームを操作するクラスが **FormApp** だったように、スプレッドシートを操作するクラスは **SpreadsheetApp** です。「Excel 方眼紙」「ネ申 Excel」などと揶揄されるように^{*7}、スプレッドシートは単なる表計算に収まらない非常に自由な表現が可能です。したがって、**SpreadsheetApp** ができる操作も非常に多く、その全てをこのテキストで紹介することはできません。本テキストでは、Slack との連携に主軸を置いているため、特にスプレッドシートから値を取り出す方法を中心に **SpreadsheetApp** の扱い方について説明します。

本節では、出席管理や、集金管理などを行うシートを想定して、以下の図 34 に示すようなスプレッドシートの値をスクリプトから取得する方法について説明します。

^{*7} ググるといろいろな議論が出てくる。



	A	B	C	D	E
1	名前	ステータス			
2	Alice	未			
3	Bob	済			
4	Charlie	済			
5	Dave	済			
6	Ellen	済			
7	Frank	済			
8					

図 34 スプレッドシート例

前節にて説明したように、スクリプトからスプレッドシートを扱う際には構造に併せて順番に参照する必要があります。したがって、スプレッドシートからある値を取り出すには、以下のような流れで処理を記述します。

1. スプレッドシートを取得する
2. スプレッドシートからシートを取得する
3. シートから範囲（または 1 つのセル）を取得する
4. 範囲（または 1 つのセル）から値を取得する

まず、スプレッドシートを取得してみましょう。

5.2.1 スプレッドシートを取得する

スプレッドシートは、Google Drive のファイル ID と同様に、**スプレッドシート ID** で一意に識別されます。したがって、スプレッドシート ID から特定のスプレッドシートを指定し、スクリプトから呼び出すことが可能です。スプレッドシート ID からスプレッドシートを取得するには、`SpreadsheetApp.openById()` を使います。

スプレッドシート ID は、スプレッドシートの URL から確認できます。スプレッドシートの URL は `https://docs.google.com/spreadsheets/d/{スプレッドシート ID}/edit` のようになっており、**{スプレッドシート ID}** の部分にスプレッドシート ID があります。

GAS のスクリプトエディタを開き、まずはスプレッドシートを取得する処理を書きます。

```

1 function spreadsheetSample() {
2   var spreadsheet = SpreadsheetApp.openById('Spreadsheet ID');
3 }

```

ソースコード 32 SpreadsheetApp sample 1

また、URL からスプレッドシート ID を抽出せずとも、URL を直接指定してスプレッドシートを取得することも可能です。その場合は、`SpreadsheetApp.openByUrl()` を使用します。

```

1 function spreadsheetSample() {
2   var spreadsheet = SpreadsheetApp.openByUrl('https://docs.google.com/spreadsheets/d/{
     Spreadsheet ID}/edit');
3 }

```

ソースコード 33 SpreadsheetApp sample 2

5.2.2 スプレッドシートからシートを取得する

前項で取得したスプレッドシートから、特定のシートを取得してみましょう。シートには名前がついています。これは、スプレッドシートにおいて一意^{*8}なので、これを使ってシートを特定し、取得できます。シート名を使ってスプレッドシートからシートを取得するには、`Spreadsheet.getSheetByName()`を使います。いま、図 34 のシートには“管理票”という名前がついていますので、以下のようなスクリプトで取得します。

```

1 function spreadsheetSample() {
2   var spreadsheet = SpreadsheetApp.openByUrl('Spreadsheet ID');
3   var sheet = spreadsheet.getSheetByName('管理票');
4 }

```

ソースコード 34 SpreadsheetApp sample 3

対象としているスプレッドシートに複数のシートがあつて、それらをまとめて取得したい場合は `Spreadsheet.getSheets()` が有効です。これは、スプレッドシートにあるシート全てを含むリストを取得することができます。

5.2.3 シートから指定した範囲のセル/値を取得する

前項で取得したシートから、指定した範囲のセル、もしくは値を取得しましょう。

セルと値の違い

あるセルに **100** と書かれていた場合を考えてみます。このセルは **100** という値以外にも、様々な情報を持っています。例えば背景色や文字色、フォントタイプ、サイズ、罫線の有無・・・といった具合です。「セル」はこれらの情報を包含する概念ですが、「値」はその「セル」がもつ **100** という値そのものを指します。

指定した範囲のセルを取得する場合は、`Sheet.getRange()`を使います。しかし、今回興味の対象はそのセルに格納されている値ですので、より簡便な `Sheet.getSheetValues()`を使います。

これら2つのメソッドは `row`, `column`, `numRows`, `numColumns` という共通する4つの引数を取ります^{*9}。これらは以下の意味を持ちます。

- **row**
取得する範囲（四角形）の左上の行番号
- **column**
取得する範囲（四角形）の左上の列番号
A 列が 1, B 列が 2,... と、整数値で指定する。

^{*8} 同じスプレッドシートに同じ名前のシートを複数作成することはできない

^{*9} 厳密には `getSheetValues` の引数名は `startRow`, `startColumn`, `numRows`, `numColumns` だが、役割はだいたい同じ。

- numRows
row 行 column 列から何行取得するか
- numColumns
row 行 column 列から何列取得するか

言葉で説明すると上記のようになるのですが、いまいち分かりづらいかと思います。そこで、以下の図 35 をベースにもう少し詳しく説明してみます。

いま、図に示したようなスプレッドシートがあり、セル B2 を左上として 5 行 4 列の範囲を取得したいとします。左上のセル B2 は上記の引数 row, column の形式に合わせるとそれぞれ 2, 2 となります。一方、取得する行数と列数はそれぞれ 5, 4 ですので、実際に `Sheet.getSheetValues(row, column, numRows, numColumns)` に与えるべき値は 2, 2, 5, 4 となります。

	A(1)	B(2)	C(3)	D(4)	E(5)
1					
2		1	2	3	4
3		5	6	7	8
4		9	10	11	12
5		13	14	15	16
6		17	18	19	20
7					

図 35 は、スプレッドシートのイメージ図です。表の行と列は、ヘッダ行（1行目）とデータ行（2行目～7行目）に分かれています。セル B2 は、行番号 2、列番号 2 の位置にあります。図には、row（2）と column（2）の値を示す矢印と、numRows（5行）と numColumns（4列）の範囲を示す括弧が描かれています。また、`sheet.getSheetValues(2, 2, 5, 4)` のコードが、取得範囲のセルに重ねて表示されています。

図 35 getSheetValues のイメージ図

さて、改めて図 34 のスプレッドシートに戻り、取得する範囲と実際に与える引数の値について考えてみましょう。1 行目はヘッダ行ですので、スクリプトが取得する必要はありません。したがって、**2 行目 1 列目**から、**6 行、2 列**取得すれば良さそうです。よって、与えるべき値は `Sheet.getSheetValues(2, 1, 6, 2)` となります。

```

1 function spreadsheetSample() {
2   var spreadsheet = SpreadsheetApp.openByUrl('Spreadsheet ID');
3   var sheet = spreadsheet.getSheetByName('管理票');
4   var values = sheet.getSheetValues(2, 1, 6, 2);
5
6   Logger.log(values);
7 }

```

ソースコード 35 SpreadsheetApp sample 4

実際、上記のスクリプトを実行すると、以下のようなログが確認でき、正しく値が取得できていることが確認できます。ここで注意しておきたいのは、実際に `Sheet.getSheetValues` に引数を与える際には行・列の番号が 1 から始まるのに対して、実際に取得されるリストのインデックスは 0 から始まる点です。1-origin と 0-origin を行き来することになりますので、少し注意が必要です。

```
1 [19-12-13 19:29:53:321 JST] [[Alice, 未], [Bob, 済], [Charlie, 済], [Dave, 済], [Ellen, 済], [Frank, 済]]
```

ソースコード 36 SpreadsheetApp output sample 1

0-origin と 1-origin

ある集合があったとき、その要素を 0 から数えるのか、それとも 1 から数えるのか、これを一般に**オリジン**と呼びます^a。

例えば、年月日はそれぞれ「元年」「1 月」「1 日」から数えますので、これは 1-origin となります。一方で、時間は「0 時」「0 分」「0 秒」から数えますので、これは 0-origin となります。

プログラミングにおいては配列・リストの添字 (インデックス) は 0-origin であることが多いです。また、JavaScript の Date クラスでは、月は 0-origin、日は 1-origin と、異なるオリジンが混在します。複数のオリジンを行き来するケースはしばしばありますので、慣れておくといいです。

^a 日本語圏で多く使われる表現であり、英語圏では zero-based, one-based と言うことが多い

さて、先程 `Sheet.getSheetValues` に引数を与えて、今値が入力されている「6 行分」の値を取得することができました。しかし、そもそも「6 行」取得しなかったのではなく、**始点の行から末尾の行まで**取得したはずです。例えば、この管理票で管理する対象の人が増えたとき、スクリプトで `getSheetValues` に与えている値をいちいち書き換えることになってしまいます。これではわざわざスクリプトを書いた意味がなくなってしまう。

そこで、取得する行数を常に自動的に取得してくれるようにします。末尾の*10行を取得するには、`Sheet.getLastRow()` を使います。図 34 の例で言うと、7 行目が末尾になりますので、このメソッドを実行すると 7 が返ってきます。

2 行目から範囲を指定していますので、実際に `Sheet.getSheetValues` に渡す際にはヘッダ行分の 1 を減ずる必要があります。したがって、`Sheet.getSheetValues(2, 1, sheet.getLastRow() - 1, 2)` となります。

したがって、最終形としては以下ようになります。このスクリプトを実行すると、先程と同様なログが出力されます。行をいくつか追加しても、正しく末尾まで取得できることを確認してみましょう。

```
1 function spreadsheetSample() {
2   var spreadsheet = SpreadsheetApp.openByUrl('Spreadsheet ID');
3   var sheet = spreadsheet.getSheetByName('管理票');
4   var values = sheet.getSheetValues(2, 1, sheet.getLastRow() - 1, 2);
5
6   Logger.log(values);
7 }
```

ソースコード 37 SpreadsheetApp sample 5

*10 値が入っているセルがある最後の行を意味する

5.3 Practice: Google Spreadsheets × Slack

本章では、スプレッドシートから指定した範囲の値を取得することを中心に、**SpreadsheetApp** の利用方法について説明してきました。ここに、Slack へのメッセージ投稿の処理を組み合わせ、簡易的なリマインダを作成してみましょう。

さきほどの管理票の名前の部分を Slack のユーザ名にかえて、ステータスが**未**の人に定期的にメンションを送る bot を作ってみてください。

ちなみに、簡単なリマインダであれば、slack の `/remind` コマンドで実現できることもあります。もし Google Spreadsheets と Google Apps Script で slack リマインダを作る場合は、`/remind` で実現可能か検討してから取り掛かったほうがよいでしょう。

5.3.1 時間主導型トリガ

定期的にスクリプトを実行させるには、**時間主導型トリガ**を使います。前章(4.2.1)で説明したとおり、トリガはスクリプトエディタの時計マークのボタンをクリックすると設定できます。

Google Forms のスクリプトの際はイベントトリガを設定しましたが、今回は時間主導型のトリガを設定します。

図 36 に示すように、「イベントのソース」を「時間主導型」とし、トリガーを設定します。「時間ベースのトリガーのタイプ」には実行間隔を設定します。実行間隔には、以下の 6 種類があります。

- 特定の日時
特定の日時に 1 度だけ実行します。スクリプトの実行予約のようなイメージです。
- 分ベースのタイマー
数分おきに実行します。間隔は以下の 5 つから選択できます。
 - 1 分おき
 - 5 分おき
 - 10 分おき
 - 15 分おき
 - 30 分おき
- 時間ベースのタイマー
数時間おきに実行します。間隔は以下の 6 つから選択できます。
 - 1 時間おき
 - 2 時間おき
 - 4 時間おき
 - 6 時間おき
 - 12 時間おき
- 日付ベースのタイマー
毎日、指定した時間帯に実行します。時間帯は、午前/午後 n 時台というように設定できます。
- 週ベースのタイマー
毎週何曜日の何時台に実行する、といった形で実行できます。

- 月ベースのタイマー

毎月何日の何時台に実行する，といった形で実行できます。

上に示したリストからわかるように，「特定の時間」に実行する方法は「特定の日時」トリガーしかなく，「特定の時間に定期的に実行する」といったトリガーの設定方法は存在しません^{*11}。これは，実際にスクリプトが実行されるサーバー側の負荷分散のための制約です。

GASxslack講座 GSSハンズオン用のトリガーを追加

実行する関数を選択

myReminder

実行するデプロイを選択

Head

イベントのソースを選択

時間主導型

時間ベースのトリガーのタイプを選択

日付ベースのタイマー

時刻を選択

午後 12 時～1 時

(GMT+09:00)

エラー通知設定

毎日通知を受け取る

キャンセル

保存

図 36 時間主導型トリガ

^{*11} ちなみに，頑張れば実現できます。see: <https://qiita.com/chihiro/items/d23692c308c89e1b1ee2>

6 Advanced Practice: Google Forms × Google Spreadsheets × Slack

ここまで、Google Forms, Google Spreadsheets を題材に、Google Apps Script の扱いかたについて説明してきました。また、練習問題を通して、ある Google Apps を Google Apps Script から操作する方法について学習しました。

発展的な課題として、Google Apps Script から、Google Forms と Google Spreadsheets のふたつを同時に操作するスクリプトを書いてみましょう。

■作成例 何らかの“既存のスプレッドシート”があったとします。そこに、フォームからの回答を転記し、Slack への通知も行うスクリプトを書いてみましょう。想定されるケースとしては、「今までスプレッドシートに直接記入してもらっていたものをフォームに置き換えた」というケースです。

7 おわりに

ちょっとしたプリント程度の分量で収まると思ったら、全然収まりませんでした。修士論文以来、都合2年ぶりに L^AT_EX でドキュメントを書いてみました。やっぱり L^AT_EX はきれいなドキュメントが簡単に作成できるのでよいですね。

さて、今回取り扱った内容は、Google Apps Script, Incoming Webhooks 共にイロハのイ程度の内容です。これらでできることはまだまだ沢山あります。一方で、どちらも公式ドキュメントが英語でしか提供されていないことが学習の障壁になっていますが、テクニカル系のドキュメントは比較的平易な英語で書かれているので、食わず（読まず）嫌いせずに読んでみましょう。ちなみに、世の中には「公式ドキュメント読めば5分で解決することで5時間も Google とにらめっこするな」という言葉があります^{*12}。

^{*12} この言葉の出典となった note の記事はリンク切れしてしまったので、類似のこちらを紹介します。『私たちはどうして公式ドキュメントが読めないのか?』（<https://qiita.com/hiraike32/items/f0a211cceb0ecc516b6c>）

付録 A 実装例

A.1 Practice: Google Forms × Slack

例えば、以下のような実装例がある。少し投稿されるメッセージの内容を作り込んでいるが、だいたいこんな感じ。なんでもかんでも `var` で宣言するんじゃないとか言わない。ていうかそもそも Google Apps Script には `let` がない^{*13}。

```
1 function gasFormPractice(eventObj) {
2   var response = eventObj.response;
3   var answers = response.getItemResponses();
4
5   var answer = answers[0].getResponse();
6
7   var email = response.getResponseEmail();
8   var mention = '<@' + email.replace('@works-hi.co.jp', '') + '>';
9
10  const WEBHOOK = '{Webhook URL}';
11
12  var text = ':secret:' + mention + ' さんの今日のお昼は【' + answer + '】らしい';
13
14  var params = {
15    'method': 'POST',
16    'headers': {'Content-type': 'application/json'},
17    'payload': JSON.stringify({'text': text})
18  }
19
20  UrlFetchApp.fetch(WEBHOOK, params)
21 }
```

ソースコード 38 Script Example for Forms practice

A.2 Practice: Google Spreadsheets × Slack

```
1 function myReminderForText() {
2
3   const ss = SpreadsheetApp.openById("Spreadsheet ID");
4   const sheet = ss.getSheetByName("管理票");
5
6   const sheetValues = sheet.getSheetValues(2, 1, sheet.getLastRow()-1, 2);
7
8   var attachments = []
9
10  for (var value in sheetValues) {
11    if (sheetValues[value][1] == '未') {
12      attachments.push({'title': 'ステータス未入力です
13      よ!', 'text': '<@'+sheetValues[value][0]+'>', 'color': '#D50200'})
14    }
15  }
16
17  const WEBHOOK = "{Webhook URL}";
18
19  var params = {
20    'method': 'POST',
```

^{*13} ランタイムを V8 にすれば `let` が使えます。


```

20     'headers': {'Content-type': 'application/json'},
21     'payload': JSON.stringify({'text': '', 'attachments': attachments})
22 }
23
24 UrlFetchApp.fetch(WEBHOOK, params)
25
26 }

```

ソースコード 39 Script Example for Spreadsheets practice

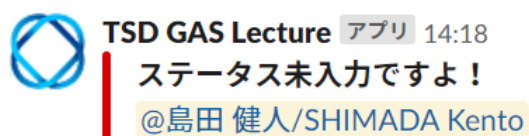


図 37 実行結果

付録 B 更新履歴

v1.0(2019/12/18): 初版公開。