詳解! Google Apps Script完全入門 第2版と第3 版の差分PDF

2020年12月に「詳解! Google Apps Script完全入門 [第2版]」が発売されました。しかし、 その前後に Google Apps Script や Google Workspace に関していくつかのアップデートがあ り、実際の環境および操作方法と、書籍のものとの間に、いくつかの差分が生まれてしま いました。

すぐに、出版社さまと相談させていただき、書籍については2021年7月に「詳解! Google Apps Script完全入門 [第3版]」を出版することで、その差分を埋めております。

しかし、「詳解! Google Apps Script完全入門 [第2版]」をお買い求めのみなさまにには、 引き続き不都合をおかけしてしまう形になってしまいます。微力ではありますが、その不 都合を補うべく、第3版で大きく書き直したパートについてPDFとして配布させていただく こととしました。

本PDFは、[第3版] の2章、20章、23章についてまとめたものです。

なお、ご利用の際には、以下についてご了承くださいませ。

- [第2版] とともにご活用いただくことを前提としております。
- [第2版] については2章、20章、23章以外の章についても、現行といくつかの差分が ございますが、それについてすべて網羅するものではないことをご了承ください。
- 出版社さまのご了承を得て、著者が独自に作成しているものとなりますため、出版社 さまには本PDFについてのお問い合わせはしないようにお願いします。

また、サンプルコードは以下出版社さまのページの「サポート」にてダウンロードが可能になる予定です。

https://www.shuwasystem.co.jp/support/7980html/6474.html

それでは、どうぞご活用いただければと思います。

2. スクリプトエディタとダッシュボード

2.1 はじめての GAS

◆スプレッドシートからスクリプトエディタを開く

スクリプトエディタは、GAS のスクリプトの編集、実行、デバッグなどを行うための専用のエディタで、メニューやツールバーをはじめ、開発を支援するための便利な機能が搭載されています。スクリプトエディタがあるおかげで、私たちは面倒な準備をすることなく、すぐに GAS をはじめることができます。

では、実際にスクリプトエディタを起動して、GASのスクリプトの作成や実行など、 GAS開発の一連の流れを確認してみましょう。

まず、Google ドライブから新規のスプレッドシートを作成します。Google ドライブで フォルダを開き、右クリックメニューから「Google スプレッドシート」を選択します(図 2-1-1)。左上の「+新規」ボタン→「Google スプレッドシート」でも同様のことが行なえ ます。

▶図 2-1-1 Google ドライブから新規スプレッドシートを作成

4	ドライブ	Q ドライブで検索		-	? 🅸		Google	P
+	新規	マイドライブ > …	> ★GAS本第3版 :	> はじめて	DGAS →	E	⊞ (i)	31
\bigtriangledown	候補							
• @	マイドライブ							0
•	共有ドライブ		新しいフォルダ					
		₽	ファイルをアップロード					
6	共有アイテム	^	フォルダをアップロード					+
\bigcirc	最近使用したアイテム	B	Google ドキュメント	>				
Δ	スター付き		Google スプレッドシート	>				
Ū	ゴミ箱	_	Google スライド	>				
		=	Google フォーム	Coo			. L	
	保存容量		その他	Good	JIEZJU	כחשי	/-r	
	801.2 GB 使用							
Ø	管理コンソール							>

ブラウザの新規タブで、作成した「無題のスプレッドシート」が開きます。次に、スプ レッドシートのメニューから、「ツール」→「スクリプトエディタ」を選択します(図 2-1-2)。

▶図 2-1-2 スプレッドシートからスクリプトエディタを開く

田 無題のスプレッドシート ファイル 編集 表示 挿入 表示形式 データ	<u>ッ</u> -			읍 共有	P
▶ 🕫 🖶 📜 100% ▾ 🕴 % .0ੵ .00 123▾	₿	フォームを作成	Α …	^	31
^A ②スクリプトエディタ	() ()	スクリプトエディタ マクロ	G	н	<mark>-</mark> Q,
3 4		スペルチェック ト			0
5 6 7		オートコンプリートを有効にする			
8 9 10		シートを保護			
11 12	Ť	ユーザー補助設定			
13		アクティビティ ダッシュボード			
+ ≡ シート1 •				Q	>

すると、ブラウザの新規タブでスクリプトエディタが起動し、「無題のプロジェクト」 が開きます (図 2-1-3)。この画面でスクリプトの編集や実行、デバッグを行っていきます。

▶図 2-1-3 スクリプトエディタ

2	Apps Script	無題のこ	プロジェクト デブロイ・ 🤇 🧿	
i	ファイル	+	5 さ 8 ▶ 実行 10 デバッグ 関数なし 実行ログ 以前のエ	ディタを使用
<>	⊐− F.gs		1 function myFunction()	
\odot	ライブラリ	+	2 3 2	
≡⊾	サービス	+	4	
٢				

◆スクリプトを編集する

では、続けてスクリプトの編集をしていきましょう。現時点では、スクリプトエディタ に以下コードが記述されているはずです。

▶サンプル 2-1-1 無題のプロジェクトのコード

```
function myFunction() {
```

}

「function myFunction() {」の行と、「}」の行の間に1行空いていますね、そこにコードを入力して、以下のサンプル2-1-2を完成させていきます。

▶サンプル 2-1-2 スクリプトの入力

```
function myFunction() {
  Browser.msgBox('Hello');
}
```

コードの入力は難しいように思われるかも知れませんが、スクリプトエディタの機能を 用いれば簡単に入力を進めることができますので、一緒に進めていきましょう。

まず、2 行目のいずれかの箇所をクリックすると、自動的に半角スペース 2 文字分だけ字 下げした位置にカーソルが置かれるのに気づくはずです。この字下げは「インデント」と いい、コードの読みやすさにとって重要な役割を果たします。そのまま字下げした位置か らコードを書き始めてください。

では、半角アルファベットの「br」と入力してみましょう。すると、図 2-1-4 のように、 入力候補が一覧表示されます。[1][4] キーで選択項目を移動できますので、「Browser」に あわせて [Tab] キーを押して確定しましょう。

2	Apps Script 無题	のプロジェクト デプロイ・ 🛛 📀 🦃
i	ז (1) ני	▶ 実行 № デバッグ 関数なし 実行ログ 以前のエディタを使用
$\langle \rangle$	O ⊐− F.gs	1 function myFunction()
\odot	ライブラリ	+ 3 Entreak
≡⊾	サービス	+ INTERFACE Browser
()		(2)Browser

続いて、ドット「.」を入力すると、再度その後の候補が一覧表示されます。「msgBox」 を選択して確定させましょう。

▶図 2-1-5 「.msgBox」の入力

2	Apps Script	無題のプロ	ジェクト	デプロイ 👻	0 🌒
i	ファイル	¢ +	①[.](ドット記号)	を入力	前のエディタを使用
$\langle \rangle$	Ο ⊐−ド.gs	1	function_myFunction() {		
Õ	ライブラリ	+ 23	Browset.		
≡⊾	サービス	+	S inputBox	(method) Browser.n	nsgB
63					
			2msgBox		

その後、開く丸括弧「(」や、シングルクォーテーション「'」を入力すると、対になる記 号が自動で入力されたりしますので活用しましょう。また、「msgBox」を説明するボック スが自動表示されますが、[Esc] キーで消すことができます。

最終的に、サンプル 2-1-2 と同じになるように進めていきましょう。なお、行最後のセミコロン「;」 忘れや、記号の打ち間違いに注意してください。

◆スクリプトを保存する

スクリプトに変更を加えると、「コード.gs」の前に黄色の丸アイコンが付与されることにお気づきでしょうか。

スプレッドシートやドキュメントなど多くの Google アプリケーションでは、クラウド上 に自動保存されることが多いのですが、**スクリプトエディタのスクリプトは自動保存がな** されません。

ですから、スクリプトに変更を加えたのであれば、都度手動で保存をする必要がありま す。

保存の方法は、図 2-1-6 に示すツールバーの「プロジェクトを保存」アイコンをクリックするか、またはショートカットキー [Ctrl] + [S] / [ม] + [S] を使用します。

▶図 2-1-6 スクリプトの保存

2	Apps Script 無題のプロジェクト	0 🌘
্র ১১	ファイル + ち こ こ (2) ノリンエクトを休存 ・ コード.gs 1 「unction myFunction()] 2 Browser.msgBox('Hello');	以前のエディタを使用
⊜⊾	サービス () 黄色の丸 アイコン	
()		

すると、黄色の丸アイコンが消えます。つまり、保存されたということです。

さて、新規作成したプロジェクトはすべて「無題のプロジェクト」という名称になって しまうので、プロジェクト名を変更しておいたほうがよいでしょう。

プロジェクト名をクリックすることで、プロジェクトの名前を変更するダイアログが開きますので、プロジェクトタイトルを入力して、「名前を変更」をクリックします(図 2-1-7)。

▶図 2-1-7 プロジェクト名の編集

2	Apps Script	無題のフ	゚ロジェク	F_	①プロジ	ェクト名		デプロイ	-	D ₀	?		9
í	ファイル	+	520	▶ 実行	行 の デバッグ	関数なし	実行ログ				以前の	エディク	マを使用
<>>	コード.gs		1 functi	on myFund	ction() {								
Ó	ライブラリ	+	2 3 } 4										
≡⊾	サービス	+		ノロジ:	エクトの名前を	変更							
()				はじる	めてのGAS	(2プロ:	ジェクト	タイトル	を入力				
					-	キャンセル 名詞	前を変更	③名前	前を変	更			

スプレッドシートから作成、保存をしたスクリプトは、再度スプレッドシートのメニュ - 「ツール」→「スクリプトエディタ」から開くことで、編集を再開できます。

◆スクリプトを実行する

では、完成したスクリプトを実行してみましょう。

ツールバーに「myFunction」が表示されているのを確認した上で、ツールバーの「▷実 行」をクリックするか、ショートカットキー [Ctrl] + [R] / [郑] + [R] を押下します(図 2-1-8)。

▶図 2-1-8 スクリプトの実行

2	Apps Script	はじめて	
i	ファイル	+	5 ♂ 🗟 🕨 実行 🛛 デバッグ myFunction 🗸 実行ログ 🛛 以前のエディタを使用
\leftrightarrow	⊐− ド.gs		1 function myFunction()
\bigcirc	ライブラリ	+	Browser.msgBox('Hello');
≡⊾	サービス	+	あることを確認
()			

(Memo)

「myFunction」は関数名といい、関数はスクリプトの実行単位です。詳しくは第 3 章および 第 5 章で解説をしますのでご覧ください。

初回の実行の際は、図 2-1-9 のように「承認が必要です」というダイアログが表示されます。

これは GAS ならではの手順ともいえるものですが、スクリプトに対して、操作対象となるスプレッドシートのアクセスを許可する必要があるのです。

▶図 2-1-9 「承認が必要です」ダイアログ

2	Apps Script	はじめてのGAS デブロイ 🝷 🛛 💿 🌒
í	ファイル	+ 5 C ■ ▶ 実行 © デバッグ myFunction ▼ □ 停止 実行ログ
$\langle \rangle$	コード.gs	承認が必要です
Ó	ライブラリ	このプロジェクトがあなたのデータへのアクセス権限を必要としています。
≡⊾	サービス	
÷		キャンセル権限を確認 ×
		権限を確認

「承認が必要です」ダイアログで「権限を確認」をクリックし、続く画面でアカウント を選択します(図 2-1-10)。

図 2-1-10 アカウントを選択

G Google にログイン

「plannauts.co.jp」 「はじ	のアカウントを選択してく ださい ^{worのGAS」に移動}
高橋宣成 ② 別のアカウントを使用	ファカウントを選択

◆アクセス許可の手順

日本語 👻

以降の手順は使用しているアカウントが Google Workspace アカウントかどうかで変わっ てきます。無料の google.com アカウントの場合は、図 2-1-11 に示す「このアプリは確認 されていません」という画面が表示されます。Google Workspace アカウントの場合は、こ の画面はスキップされますので、次の手順として図 2-1-13 へ進んでください。

ヘルプ プライバシー 規約

「このアプリは確認されていません」の画面では「詳細」をクリックしてください。 「安全なページに戻る」をクリックすると、アクセス許可されずに戻ってしまいます。



すると、図 2-1-12 のように画面下部に「はじめての GAS(安全ではないページ)に移動」というリンクが表示されますのでクリックします。

▶図 2-1-12 安全ではないページに移動

このアプリは Google で確認されて	いません
アプリが、Google アカウントのプライベートな情報への す。デベロッパー()とG れるまで、このアプリを使用しないでください。	Dアクセスを求めていま oogle によって確認さ
詳細を非表示	安全なページに戻る
リスクを理解し、デベロッパー(場合のみ、続行してください。)を信頼できる
はじめてのGAS (安全ではないページ) に移動	
はじめてのGAS(安全ではた に移動	ないページ)

続いて、図 2-1-13 のように「はじめての GAS が Google アカウントへのアクセスをリク エストしています」という画面に移りますので、「許可」をクリックしてアクセスを許可 しましょう。

▶図 2-1-13 スクリプトにアクセスを許可

G Google にログイン はじめてのGAS が Google アカウントへの アクセスをリクエストしています **(** はじめてのGAS に以下を許可します: Google ドライブのスプレッドシートの表示、編集、作成、 (;) 削除 はじめてのGAS を信頼できることを確認 機密情報をこのサイトやアプリと共有する場合があります。 はじめ てのGAS の利用規約とプライバシー ポリシーで、ユーザーのデータ がどのように取り扱われるかをご確認ください。 アクセス権の確 認、削除は、Google アカウントでいつでも行えます。 リスクの詳細 許可 キャンセル 許可

許可が完了すると、スクリプトの実行がなされます。スクリプトエディタでは何も起き ていないように見えますが、実はちゃんと動いています。スプレッドシートに画面を切り 替えると、図 2-1-14 のように「Hello」と表示されたメッセージボックスが表示されている はずです。

▶図 2-1-14 メッセージボックスの表示

	はじめて ファイル	のGAS ☆ 編集 表示 挿	D ② ス 表示形式 デ	ータ ツール	アドオン へノ	レプ 量…)~ 9	島 共有	9
ĸ		100% - ¥	% .0 <u>,</u> .0 <u>0</u> 12	23 - デフォル	ト → 10	- B I S	<u>A</u>	^	31
fx									
	A	В	С	D	E	F	G	н	
1]							
2						×			
3						~			•
4			Hello						
5									
6						OK			
7						OK			
8									
9									
10									
11									
12									
13									
								★ >	
	+ ≣ ೨	/−ト1 -							>

これでスクリプトエディタの起動、スクリプトの編集、保存、実行と、GASの一連の流れを確認することができました。しかし、スクリプトエディタには、ここでお伝えした以外にも、GASの開発や習得に便利な機能がたくさん搭載されています。

以降の節で、実際にそれらスクリプトエディタの機能に触れながら、GAS の仕組みの理 解を深めていくことにしましょう。

2.2 プロジェクトとスクリプト

<u>◆プロジェクトとファイル</u>

GAS のスクリプトは、プロジェクトという単位で作成します。プロジェクトは単一また は複数のファイルで構成されており、GAS のスクリプトは拡張子「.gs」のファイルに記述 をします (図 2-2-1)。また、プロジェクト内に「.html」ファイルを作成することもできま す。

▶図 2-2-1 プロジェクトとファイル



プロジェクト

図 2-2-2 は、新規作成した時点のプロジェクトをスクリプトエディタで開いた画面で す。「無題のプロジェクト」がプロジェクト名、プロジェクトは「コード.gs」という1つ のファイルで構成されています。

スクリプトエディタメニューの「ファイル」の「+」アイコンから新たなスクリプトフ ァイル (.gs) および HTML ファイル (.html) を作成できます。スクリプトファイルを新規 作成すると、ファイル一覧にファイルが追加されます。ファイル一覧でファイルをクリッ クすると、その内容が入力エリアに表示され、編集ができるようになります(図 2-2-3)。

▶図 2-2-2 新規作成した時点のスクリプト

2	aff Apps Script (無題のプロジェクト) デプロイ マ ⑦ ◎					
() <>	ファイル + s e ■ ▶ 実行 □-F.gs Jロジェクト名 ヷ					
Õ	ライブラリ + 1 function myFunction()					
≡⊧	* ファイル一覧					
\$						

▶図 2-2-3 複数のファイルで構成されるプロジェクト

2	Apps Script	無題のプロジェクト デブロイ・ ⑦ 👰
i	ファイル	ー
<>	⊐− ド.gs	以前のエディタを使用
Õ	追加.gs	1 function myFunction()
≡⊾	5175	+ 2
()	ファイノ	レを選択

<u>◆スタンドアロンスクリプトとコンテナバインドスクリプト</u>

GAS のスクリプトには 2 種類あり、どちらの種類を使うかで作成および保存の方法が異なります。

1つは、**スタンドアロンスクリプト**といい、Google ドライブにプロジェクトファイル自体を直接保存します(図 2-2-4)。こちらは、プロジェクトファイル1つに対し、1つのプロジェクトしか持つことができません。

▶図 2-2-4 スタンドアロンスクリプト



もう1つはスプレッドシート、ドキュメント、フォームおよびスライドといった、親フ ァイル(**コンテナ**といいます)に紐づく形で保存するコンテナバインドスクリプトです (図2-2-5)。なお、2-1 で作成および実行をしたスクリプトは、スプレッドシートに紐づ いている**コンテナバインドスクリプト**です。

▶図 2-2-5 コンテナバインドスクリプト



(Memo)

コンテナバインドスクリプトは、スプレッドシート、ドキュメント、フォームおよびスライドで作成できます。

両者の比較について、図 2-2-6 にまとめています。ここに記載した内容以外の点では、 どちらも同じ動作をします。

▶図 2-2-6 スタンドアロンとコンテナバインド

	スタンドアロンスクリプト	コンテナバインドスクリプト
スクリプト の保存	単体のスクリプトプロジェク トファイルとして保存	スプレッドシート、ドキュメント、フ ォーム、スライドに紐づいて保存
スクリプト の作成	Apps Sciprtタッシュボードま たはGoogleドライブから作成	スプレッドシート、ドキュメント、フ ォーム、スライドのメニューから作成
スクリプト のオープン	Apps Scriptダッシュボードま たはGoogleドライブからプロ ジェクトファイルを開く	スプレッドシート、ドキュメント、フ ォーム、スライドのメニューから開く
メソッド		紐づいたファイルに対するいくつかの メソッドが利用可能
トリガー	時間主導型、HTTPリクエス ト、カレンダーからのトリガ ーを設置可能	時間主導型、HTTPリクエスト、カレン ダーからのトリガーに加えてコンテナ に関するトリガーを設置可能
UI	_	コンテナのUIのカスタマイズが可能
スプレッド シートのカ スタム関数	_	作成可能
ライブラリ の作成	作成可能	_

コンテナバインドスクリプトでは、スプレッドシートのカスタム関数の作成ができる、 また紐づいたファイルを参照する getActiveSpreadsheet、getActiveDocument、getUi などの、いくつかの便利なメソッドを使用することができるといったメリットがあります。

(Memo)

基本的に、GAS によるツールを作成するとき、コンテナバインドスクリプトをメインに使用していくという方針でよいでしょう。しかし、ライブラリの作成など、コンテナバインドスクリプトでは都合がよくない場合もありますので、その際にスタンドアロンを使用するケースが出てきます。

◆Apps Script ダッシュボードとは

コンテナバインドスクリプトはコンテナに紐付いて保存されていますが、Google ドライ ブの表面上からはスクリプトが存在してるかどうか、またそのプロジェクト名は何かとい ったことを把握することができません。

そこで、登場するのが **Apps Script ダッシュボード**です。Apps Script ダッシュボード は、GAS のプロジェクトを管理およびモニタリングするためのツールで、以下のようなこ とを行うことができます。

- プロジェクトの閲覧および検索、操作
- スクリプトの使用状況のモニタリング
- インストーラブルトリガーの作成、管理
- スタンドアロンスクリプトの作成
- ログの表示と保管

(Memo)

上記のうち、個別のプロジェクトに対する使用状況のモニタリング、ログの表示と保管、インストー ラブルトリガーの作成、管理であれば、スクリプトエディタからも行うことができます。

コンテナバインドスクリプトもスタンドアロンスクリプトもどちらも合わせて管理でき ますので、直接的にプロジェクトを探して開いたり、確認したりするときに便利です。

Apps Script ダッシュボードは、図 2-2-7 のようにスクリプトエディタの左上のアイコン からアクセスできます。

▶図 2-2-7 Apps Script ダッシュボードを開く

2	Apps Script ダッシュボードへ ^② 🦃					
í	ファイル					
$\langle \rangle$	⊐− ド.gs		以前のエディタを使用			
Õ	ライブラリ	+	1 function myFunction()			
≡⊾	サービス	+	2 3) 4			
1						

または、以下の URL で直接開くこともできますので、ブックマークしておくとよいでしょう。

図 2-2-8 が Apps Script ダッシュボードの画面です。

左側のメニューで表示を切り替えたり、上部の検索ボックスでプロジェクトの検索を行うことができます。中央のプロジェクトのリストにはプロジェクトの一覧が表示されていて、編集アイコンでプロジェクトを開いたり、三点リーダーアイコンのメニューからスターを付ける、削除するといった操作を行うことができます。

▶図 2-2-8 Apps Script ダッシュボードの画面構成



プロジェクトをクリックすると、図 2-2-9 のプロジェクトの詳細画面が表示されます。 ここで、スクリプトの過去 7 日間の実行状況を確認できます。右側の「プロジェクトの詳 細」ではプロジェクトの情報を確認するとともに、プロジェクトやコンテナを開くことも 可能です。

▶図 2-2-9 プロジェクトの詳細



このように、Apps Script ダッシュボードは GAS のプロジェクトを管理する上で、強力な ツールになりますので、ぜひご活用ください。

◆スタンドアロンスクリプトの作成と開き方

スタンドアロンスクリプトを作成する場合は、Apps Script ダッシュボードから作成する のが便利です。Apps Script ダッシュボードの左上の「+ 新しいプロジェクト」ボタンをク リックします(図 2-2-10)。

▶図 2-2-10 新しいプロジェクトを作成する

	Q、 プロジェクト名を検索			III 🧔
➡ 新しいプロジェクト	自分のプロジェクト	多数の	うち 50 個のプロジェク	~を表示しています
☆ スター付きのプロジェクト	新しいプロジェクト			
自分のプロジェクト	プロジェクト	オーナー	最終更新	
すべてのプロジェクト	🎁 はじめてのGAS 🚢 はじめてのGAS	自分	10:38	/ E
2、 共有済み	- 102章:サンプルスクリプト 🚢	自分	2020/09/09	
 ゴミ箱 	🏠 21章:サンプルスクリプト 🚢	自分	2020/09/09	
(…) 実行数	🏠 ノンプロ研中級講座GASコース(V8) 🚢	自分	2020/09/09	
◎ マイトリガー	🔿 MyLibrary 🚢	自分	2020/09/08	
(+1 ⁿ .xh) ⁻	🏠 フォームテスト 🏭	自分	2020/09/07	

これだけでスタンドアロンスクリプトを作成できます。別のタブで、図 2-2-11 のように スクリプトエディタが開きますので、すぐに編集を開始できます。

▶図 2-2-11 作成したスタンドアロンスクリプト

🔐 Apps Script 🛛 無題の		無題	カプロジェクト 🛛 🗾 🚽 📀 🌘
i	ファイル	+	5 c ┃ ■ ▶ 実行 № デバッグ 関数なし 実行ログ
<>	⊐− ド.gs		以前のエディタを使用
Í	ライブラリ	+	1 function myFunction()
≡⊾	サービス	+	3
(3)			

作成したプロジェクトは、図 2-2-12 のように Apps Script ダッシュボードの一覧に追加 されますので、クリックすることで再度開くことができます。

▶図 2-2-12 スタンドアロンスクリプトを開く

■ Google Apps Script	Q、 プロジェクト名を検索		··· 🦃
➡ 新しいプロジェクト	自分のプロジェクト	多数のうち 50 個のプロジ	ェクトを表示しています
	+ フィルタを追加		
☆ スター付きのプロジェクト	プロジェクト	オーナー 最終更新	
自分のプロジェクト		21 2 40010 50.001	
すべてのプロジェクト	🗔 スライドイベント 🚢	自分 2020/09/07	
2、 共有済み	● 無題のプロジェクト	自分 2020/09/04	1 :
□ ゴミ箱	23章:サンプルスクリプト	プロジェクトをクリック	
	→ MyLibrary	日万 2020/09/04	
(…) 実行数			
◎ マイトリガー	🄁 22章:サンプルスクリプト 🚢	自分 2020/09/02	
	▶ 20章:サンプルスクリプト 🚢	自分 2020/09/02	
🕑 はじめに	🏠 10章:サンプルスクリプト 🎎	自分 2020/09/01	

なお、このように作成したスタンドアロンスクリプトは、Google ドライブ上ではマイド ライブに作成されます。必要に応じて、適切なフォルダに移動してください。

2.3. スクリプトエディタの編集機能

♦候補の表示

GAS で使用するオブジェクトやメソッドには、「つづり」の長いワードも多く存在しています。それらのワードを手打ちで入力することはタイプミスにつながりますし、そもそも、ワードのつづりを覚えることが難しいでしょう。

また、GAS ではアルファベットの大文字と小文字が区別されますので、

「Browser.msgBox('Hello')」を入力すべきときに、「browser」や「msgbox」などとするだ けでエラーとなります。

(Point)

GAS では、大文字と小文字を明確に区別するので注意が必要です。

しかし、そのような GAS での入力を支援するために、スクリプトエディタには**候補の表 示**という機能があります。この機能は、2-1 ですでに体験いただいていますが、再度おさら いをしておきましょう。

スクリプトエディタを開き、空白行で [Ctrl] + [Space] キーを入力すると、図 2-3-1 のように候補のリストが表示されますね。

(Memo)

Mac や Chromebook をお使いの場合は [Ctrl] + [Space] キーが他に割り当てられてい て候補の表示が使えないときがあります。そのような場合は、 [Ctrl] + [Space] キーを長押 ししたり、以下のように [fn] や [option] および [Alt] などと組み合わせると機能するこ とが多いので試してみてください。

- Mac: [fn] + [control] + [space] / [control] + [\mathbb{H}] + [space] / [fn] + [control] + [\mathbb{M}] + [space] - Chromebook: [Ctrl] + [Alt] + [Space]

ここに表示されている候補は、トップレベルオブジェクトや予約語などとよばれるもの です。GAS のコードはこれらのキーワードのいずれかを使って書き始めることになりま す。[1][1] キーで選択して、 [Tab] キーまたは [Enter] キーで入力を確定できます。もちろ ん、アルファベットの大文字、小文字も正しく入力がされます。

▶図 2-3-1 候補の表示



(Memo)

これを機に、トップレベルオブジェクトや予約語にひととおり目を通しておくのもよいでしょう。GAS の 全体像を垣間見ることができます。

2-1 で体験いただいたとおり、候補は文字を入力するだけで自動表示されます。さらに、 文字入力を進めていくと、図 2-3-2 のように候補が絞り込まれていきます。

▶図 2-3-2 候補の絞り込み

2	🔐 Apps Script 無題のプロジェクト 🗾 🚽 🕜 🁰				
i	ファイル	+	5 ♂ 記 ▶ 実行 IO デバッグ 関数なし 実行ログ		
<>	<mark>○ ⊐−</mark> ド.gs		以前のエディタを使用		
$\overline{\mathbb{O}}$	ライブラリ	+	1 function myFunction()		
≡⊾	サービス	+	2 bro 3 } // Browser interface Browser		
*			候補のリスト		

また、すでに定義されている変数・定数名や関数名も候補として表示されます。たとえば、図 2-3-3の状態で「n」と入力すると、その候補として「numA」や「numB」が含まれていることが確認できます。

▶図 2-3-3 変数名の候補表示

2	🔐 Apps Script 無題のプロジェクト 🗾 🚽 🧿 🁰				
í	ファイル	+	5 ♂ 🗟 ▶ 実行 Ю デバッグ 関数なし 実行ログ		
\leftrightarrow	O ⊐−ド.gs		以前のエディタを使用		
Ó	ライブラリ	+	1 function myFunction() ② 复数も候補に		
≡⊾	サービス	+	2 let numB; 4 pH		
ŝ			5 6 P numA let numA: any P numB ≅ null [♥] Number		

(Memo)

変数、定数については第3章で、関数については第5章で紹介します。

[Point]

候補の表示は、GAS の入力作業の強い味方になりますので、確実に使えるようにしておきましょう。

◆検索と置換

スクリプトの編集時に、[Ctrl] + [F] / [X] + [F] キーで、検索バーを表示できます。この 検索バーの入力欄にキーワードを入力することで検索を行うことができます(図 2-3-4)。 検索にマッチした箇所がオレンジ色に、その行が黄色にハイライトされます。複数の検索 結果がある場合は、[Enter] キーで次の検索結果に、 [Shift] + [Enter] キーで前の検索結果に 移動できます。

▶図 2-3-4 検索とその機能

2	Apps Script	無題のプロ語	ジェクト	デプロイ 🔻	완 🕐 🌘
(i) ()	ファイル 9-7-5-6	+ 5 さ 以前のエ		牛の設定	実行ログ
Ó	検索1	ミーワード	let	Aa <u>Abi</u> _* 1 / 2 件	$\uparrow \downarrow = \times$
≡, ⊛	サービス	+ 1 fu 2 3	<pre>inction myFunction() { let numA; let numB;</pre>	検索	々ツール
		4 } 5			

検索バーの右側に配置する4つのアイコンは、左から順番に、検索に関する以下のよう な操作を行うものです。

- 前の検索結果
- 次の検索結果
- 選択範囲を検索
- 閉じる

また、キーワード入力欄内の右側の3つのアイコンは、検索条件を設定するもので、左から順番に以下のような設定の切り替えを行います。

- 大文字と小文字を区別する
- 単語単位で検索する
- 正規表現を使用する

検索バーの表示中に「>」アイコンをクリックすると、検索バーが置換モードに切り替わります(図 2-3-5)。または、検索バーの非表示時も含めて [Ctrl] + [H] / [出] + [H] キーで置換の機能を呼び出すことができます。

▶図 2-3-5 置換とその機能

2	Apps Script 無題の	カプロジェクト 🗾 🚽 💿 🌘
() () ()	ファイル + コード.gs ライブ 置換モー	ち 置換後キーワード トマーン 実行ログ トマーン またい トロン またい トマーン またい トマーン またい トマーン またい トマーン またい トロン またい
= , ŵ	Ψ-Ex +	<pre>1 function myFunction() { 2 let numA; 3 let numB; 4 } 5</pre>

下の入力欄に置換後のキーワードを入力して、[Enter] キーで現在の検索結果を置換、 [Ctrl]+ [Alt] + [Enter] / [ង] + [Enter] キーですべての検索結果を置換します。置換の操作 は、置換後キーワードの入力欄の右側のアイコンでも実行可能です。

◆「元に戻す」と「やり直す」

スクリプトエディタでは、たとえ直前の動作の後にファイルの保存をしてしまったとしても、直前の操作を取り消し、元の状態に戻すことができます。その方法は、ツールバーの「**元に戻す**」ボタン、またはショートカットキー [Ctrl] + [Z] / [郑] + [Z] です。

また、ツールバーの「**やり直す**」ボタン、またはショートカットキー [Ctrl] + [Y] / [#] + [Y] で、元に戻した操作をやり直すこともできます。それぞれのアイコンの位置は、図 2-3-6 をご覧ください。

▶図 2-3-6 「元に戻す」と「やり直す」

2	Apps Script	無題の	っプロ・元に戻す/やり直す 🎽 🖉 🌘
í	ファイル	+	5 C ■ ▶ 美行 D テハック myFunction ▼ 実行ログ
<>	⊐−ド.gs		以前のエディタを使用
\odot	ライブラリ	+	1 function myFunction()
≡⊾	サービス	+	2 let numA; 3 let numB; 4 N
()			- <u>u</u> 5

◆スクリプトエディタのショートカットキー

スクリプトエディタには、これまで紹介したものも含めて便利なショートカットキーが 用意されていますので、図 2-3-7 にまとめています。いずれも GAS の開発効率を上げる有 用なものばかりですので、使いこなせるようにしておくとよいでしょう。

▶図 2-3-7 スクリプトエディタのショートカットキー

メニュー	操作	Windows	Мас
ファイル	ファイルを保存	[Ctrl] + [S]	[壯] + [S]
編集	元に戻す	[Ctrl] + [Z]	[쁐] + [Z]
編集	やり直す	[Ctrl] + [Y]	[쁐] + [Y]
編集	検索	[Ctrl] + [F]	[跆] + [F]
編集	置換	[Ctrl] + [H]	[쁐] + [H]
編集	すべての出現箇所を変更	[Ctrl] + [F2]	[shift] + [跆] + [L]
実行	選択している関数を実行	[Ctrl] + [R]	[郑] + [R]
	単語単位で移動	[Ctrl] + [←][→]	[option] + [←][→]

メニュー	操作	Windows	Мас
移動・削除・コ ピー	定義へ移動	[Ctrl] + [F12]	[郑] + [F12]
移動・削除・コ ピー	行の移動	[Alt] + [↑][↓]	[option] + [↑][↓]
移動・削除・コ ピー	行を削除	[Ctrl] + [Shift] + [K]	[shift] + [跆] + [K]
移動・削除・コ ピー	行のコピー	[Shift] + [Alt] + [↑][↓]	[shift] + [option] + [↑][↓]
表示	候補をトリガー	[Ctrl] + [Space]	[control] + [space]
表示	コマンドパレットの表示	[F1]	[F1]
表示	コンテキストメニューの 表示	[Shift] + [F10]	[shift] + [F10]
整形・コメント	インデントをする	[Tab]	[tab]
整形・コメント	ドキュメントのフォーマ ット	[Shift] + [Alt] + [F]	[shift] + [option] + [F]
整形・コメント	行コメントの切り替え	[Ctrl] + [/]	[郑] + [/]
整形・コメント	ブロックコメントの切り 替え	[Shift] + [Alt] + [A]	[shift] + [option] + ΓΑ]

2.4. ログとデバッグ

◆ログの出力

GAS では、スクリプトの動作を確認するためログを出力する機能が用意されています。 以下のサンプル 2-4-1 を入力して、実行をしてみてください。

トサンプル 2-4-1 ログ出力するスクリプト [sample02-04.gs]

```
function myFunction02_04_01() {
  console.log('Hello GAS!');
  console.log('I am', 25, 'years old.');
}
```

実行すると、図 2-4-1 のように「実行ログ」という画面が表示され、その一部に「Hello GAS!」「I am 25 years old.」という表示が確認できます。

▶図 2-4-1 実行ログ

2	Apps Script	02章	Ē:サンプルスクリプト 🛛 🗾 🗇 🌘
(i)	ファイル 02_01.as	+	 5 さ □ ▶ 実行 ② デバッグ myFunction02_04_01 ▼ 実行ログ 以前のエディタを使用
0	02_03.gs		<pre>1 function myFunction02_04_01() { 2 console.log('Hello GAS!'):</pre>
≡. ⊛	ライブラリ	+	<pre>console.log('I am', 25, 'years old.'); }</pre>
	サービス	+	実行ログ ×
			16:49:35 みないつせ 実行頂短い 16:49:36 情報 Hello GAS!
			16:49:36 「備報紙 I am 25 years old. 16:49:36 お知らせ 実行完了

ここで登場する「console.log」は、GAS でログ出力をするための命令です。以下のよう に、括弧内に複数の値をカンマ区切りで記述することにより、それらの値をログに出力で きます。

▶構文

console.log(値1[, 値2, …])

なお、構文内の角括弧は省略可能という意味です。以降の構文でもこの表記法が登場しますので、覚えておきましょう。

(Memo)

console クラスは GAS の Base サービスで提供されているクラスです。

実行ログでは、最新の実行についてのログのみを確認できますが、左側にあるメニューの「**実行数**」から、ログを表示したい実行をクリックすることで、過去のログを確認できます(図2-4-2)。

▶図 2-4-2 実行数

2	Apps Sc	ript 02章:サ	ンプルスク	リプト	デプロイ	• 0	P
í	実行数	Į.		過去7日間の3個実行	を表示中 リア	ルタイムで表示: 🧲	
<>	1)3	実行数					
	導入	関数	種類	開始時間	期間	ステータス	
ŵ	Head	myFunction02_04_01	エディタ	2021/01/26 16:56:03	0.365 秒	完了	
	Head	myFunction02_04_01	エディタ	2021/01/26 16:55:55	0.362秒	完了 : ^	כ
	Cloud Ø)ログ	-"" 42	27	長示した	とい実行	
	2021	/01/26 16:55:55	デバッグ Iam	25 years old.		更新	
	1ページま	ちたりの行数: 50 👻			< ^-	-ジ1/1 < >	,

「実行数」の画面では、過去7日分の実行結果とそのログが蓄積されています。また、 トリガーなどにより、スクリプトエディタを開かずに実行したものも確認できます。 (Memo)

ログとその出力については、第 16 章でも詳しく紹介していますので、そちらもご覧ください。

◆デバッグ機能

スクリプトエディタには、デバッグ機能が搭載されています。ツールバーの「デバッ グ」をクリックすると、選択されている関数についてデバッグ実行を行います。

図 2-4-3 のように、スクリプトの行番号の左側をクリックすることで、**ブレークポイン** トを設置できます。この状態でデバッグ実行を行うと、各ブレークポイントの位置で一時 停止します。

▶図 2-4-3 ブレークポイントの設置とデバッグ実行



ー時停止中は、画面右側に表示される**デバッガ**を使って、停止している時点の情報を確認したり、デバッグ実行の操作を行うことができます(図 2-4-4)。

2	Apps Script	02章: 1	サンプルスクリプト デプロイ・	0	
(j)	ファイル	+ 5	e l レオ デバッグツール し デバ	ソガ 🕐	×
<>>	02_01.gs 02_03.gs	実行		? : :	
_ ■	02_04.gs	1 2 • 3	function myFunction() let num; num = 100; コールスタック	レスタック nction @ 02_04	1:5
567	ライブラリ	$+$ $^{4}_{\circ 5}$	num = num + 100 console.log(num);		
622	サービス	$+$ $\frac{6}{7}$) 変数	_	
			v Loca nu) Glob 変数	il n: 200 ial	

デバッグツールを使って、デバッグ実行の方法をコントロールできます。それぞれのア イコンは、左側から順に以下の機能となります。

- 再開: デバッグ実行を再開する
- ステップオーバー:1行ずつ実行、呼び出した関数では一時停止しない
- ステップイン:1行ずつ実行、呼び出した関数でも一時停止する
- ステップアウト:現在の関数の最後まで実行する

デバッガでの「変数」には、停止時点に使用されている変数とその内容が表示されま す。また、コールスタックでは、停止位置のファイル名と行数とともに、関数の呼び出し 履歴を確認できます。

これら、デバッグ実行とデバッガの機能は、スクリプトの動作確認や、不具合(バグといいます)を解決するときに大いに役立ちますので、ぜひ使いこなしていきましょう。

◆スクリプトの共有と権限

GAS はクラウドにスクリプトが存在していますから、どのユーザーが該当のスクリプト の編集や実行についての権限を持っているのかが明確に定められています。図2-5-1に示す 通り、スクリプトの権限のレベルは3段階に分かれています。

▶⊠	2-5-1	スクリ	ワト	~の権限
----	-------	-----	----	------

操作	オーナー	編集者	閲覧者
スクリプトの閲覧	0	0	0
実行・デバッグ	0	0	0
ログ表示	0	0	0
トリガー設定	0	0	0
スクリプトの編集	0	0	
他のユーザーの権限設定	0	0	
プロジェクトの公開	0		
オーナー権限の譲渡	0		

まず、スタンドアロンスクリプトの場合は、スクリプトの作成者がオーナーとなりま す。他のユーザーと共有するには、図 2-5-2 のスクリプトエディタの右上の「他のユーザ ーとこのプロジェクトを共有」アイコンをクリックして、「ユーザーやグループと共有」 ダイアログを開きます。



続いて、入力欄に共有したいユーザーの名前かメールアドレスを入力します(図 2-5-3)。

▶図 2-5-3 ユーザーやグループと共有

App	≥ ユーザーやグループと共有	÷	0 ()
() □- () □- () ⋽-	ユーザーの名 高橋宣成 (自分)	前または スを入力	
=, ^{サ-}	<u>Google にフィードバックを送信</u> 保留に	中の変更保存	
		: リンクをコピー	

すると、図 2-5-4 の画面となります。ここで、共有するユーザーの権限をプルダウンか ら選択し、「送信」をクリックすることで共有が完了です。「通知」にチェックが入って いる場合は、共有先のユーザーにメールで通知されますので、必要に応じてチェックおよ びメッセージを入力するとよいでしょう。

▶図 2-5-2 スタンドアロンスクリプトの共有

▶図 2-5-4 ユーザーの権限の選択と送信

🄐 Арр	 く ユーザーやグループと共有 	3
	● a 橋 a 成 × ①ユーザーの権限 編集者 ・	
<u>o</u> =-	☑通知 を選択 閲覧者	
=, サ-	メッセージ イ 編集者	
÷		
	Google にフィードバックを送信 キャンセル 送信	

コンテナバインドスクリプトの権限は、親ファイルとなるコンテナの権限と紐づいてい ます。

ですから、コンテナの作成者がそのままスクリプトのオーナーとなります。他のユーザ ーと共有する場合は、図 2-5-5 のようにコンテナの「共有」から共有の操作をすること で、スクリプトの共有も同時になされます。以降の共有の手順はスタンドアロンスクリプ トの手順と同様です。

▶図 2-5-5 コンテナバインドスクリプトの共有



(Memo)

コンテナの権限が「閲覧者」または「閲覧者(コメント可))」である場合、コンテナのメニュー で「ツール」→「スクリプトエディタ」が表示されません。しかし、Apps Script ダッシュボードから開く ことで閲覧可能です。

共有について覚えておくべき重要な点として、異なるドメインどうしではコンテナ(そのコンテナバインドスクリプトも含む)およびスタンドアロンスクリプトのオーナー権限の譲渡ができません。したがって、組織で Google Workspace を使用している場合で、外部のパートナーと共同で GAS の開発や運用をする際には注意をする必要があります。

【Point】 スクリプトのオーナー権限は異なるドメイン間では譲渡ができません。

◆スクリプトからのアクセス許可

本章の冒頭で「はじめての GAS」の初回実行時に、「承認が必要です」ダイアログが表示されたのを覚えていますでしょうか?

その際は、スクリプトに対してスプレッドシートへのアクセス許可を与えましたね。GAS のスクリプトの多くは Google アプリケーションにアクセスをしますから、**スクリプトがそ**

れぞれのアプリケーションにアクセスをしてもよいという許可を与える必要があるのです。

GAS ではスクリプトを実行する、またはトリガーを設定する際に、自動的にコードの内 容を検査し、必要となるアプリケーションへの許可を求めにいきます。許可はプロジェク ト単位でなされ、一度与えられた許可は解除をするまで保持されます。

[Point]

スクリプトから Google アプリケーションを操作するには、それぞれのアプリケーションへのアクセス許可が必要です。

(Memo)

アクセス許可の情報は、スクリプトエディタの左メニュー「概要」を開き、プロジェクトの詳細画面の「プロジェクトの OAuth スコープ」の欄で確認できます。

2.6. サポートメニューとリファレンス

<u>◆サポートメニュー</u>

スクリプトエディタの右上の「?」アイコンは**サポートメニュー**です(図 2-6-1)。

▶図 2-6-1 サポートメニュー

2	Apps Script	02章	É: サンプルスクリプト	ナポートメニュー
i	ファイル	+	ち ♂ 🗟 ▷ 実行 🖸 デバッグ myFun	ドキュメント グ
$\langle \rangle$	02_01.gs		以前のエディタを使用	トレーニング
$\tilde{\mathcal{O}}$	02_03.gs		1 function mvFunction02 04 01() {	再 新
=	02_04.gs		<pre>2 console.log('Hello GAS!'); 3 console.log('I am' 25 'years old </pre>	史和
_►	ライブラリ	+	4 } 5	利用規約
3	サービス	+	6 // 2-4 7 // function myFunction() {	
			<pre>8 // let num; 9 // num = 100; 10 // num = num + 100 11 // console.log(num); 12 // } 13</pre>	フィードバックの送信

メニューの上から4つについては、以下公式ドキュメントへのリンクとなっています。

- ドキュメント: 公式トップ https://developers.google.com/apps-script/
- トレーニング: Samples https://developers.google.com/apps-script/articles/
- 更新: Release Notes https://developers.google.com/apps-script/releases/
- 利用規約: Additional Terms https://developers.google.com/apps-script/terms

また、「フィードバックの送信」では、スクリプトエディタの不具合の報告や、要望の 送信が可能です。

◆リファレンスの活用

スクリプトを作成する上で、GAS で提供されているたくさんのクラスやメソッドについ て知る必要があります。スクリプトエディタ上でもそれらの情報を得ることができます が、より詳しく調べたいときには、公式ドキュメントの「Reference」を使うことができま す。

サポートメニューの「ドキュメント」から公式ドキュメントを開き「Reference」タブを 選択するか、以下 URL で開くことができます。

Reference overview | Apps Script

https://developers.google.com/apps-script/reference

リファレンスページの左側の一覧から、調べたいサービスやクラスについてアクセスで きます(図 2-6-2)。

▶図 2-6-2 リファレンス:サービス概要



各メソッドの解説は、図 2-6-3 のようになっています。基本構成はすべてのメソッドで 統一されていますので、構成さえ理解してしまえば英語が苦手でも読み進めやすくなると 思います。

▶図 2-6-3 リファレンス:メソッド

I Google Apps Script				Q Search	English	- (j)	^
Overview Manifests Manifest structure Google Workspace Add-ons Dependencies resource Shears converge	createEvent Creates a new ev If no time zone is than the calenda	(title, vent. s specified, th r's time zone	startTime, endTime, options) ne time values are interpreted in the context of the script	トンプルコード	T. N D d	ible of content ethods etailed comentation createAllDayEt (title, date) createAllDayEt (title startDat	ven Te
Google Workspace services Calendar Overview CalendarApp Classes Calendar	// Creates : var event = new Date new Date locatio Logger.log(an event fo CalendarAp e('July 20, e('July 20, on: 'The Mo 'Event ID:	<pre>pr the moon landing and logs the ID. p.getDefaultCalendar().createEvent('Apollo 11 Landing' 1969 20:00:00 UTC'), 1969 21:00:00 UTC'), son')); ' + event.getId());</pre>	₽ ₪		endDate) createAllDayEt (title, startDat endDate, options) createAllDayEt (title, date, options) createAllDayE tSeries(title, startDate,	ven ven ven
CalendarEvent CalendarEventSeries EventGuest EventRecurrence RecurrenceRv	Parameters Name	Туре	Description			recurrence) createAllDayE tSeries(title, startDate, recurrence, options)	ven
Enums Color EventColor GuestStatus	title startTime endTime	String Date Date	the title of the event the date and time when the event starts the date and time when the event ends			createEvent(tit startTime, endTime) createEvent(tit startTime, endTime, optione)	tle, tle,
Visibility Contacts Data Studio Document	options Advanced param	object	a Javascript object that specifies advanced parameters, as listed belo	~		createEventFro Description(de ription) createEventSe s(title, startTir codTire	om HSC Prie The,
追加の パラメーター	Name description location quests	Type String String String	Description the description of the event the location of the event a comma-separated list of email addresses that should be added a	s quests		recurrence) createEventSe s(title, startTin endTime, recurrence, options) deleteCalenda	rrie ne, ar()
 Maps Sites Sites Spreadsheet Advanced Google services 	sendInvites Return CalendarEvent	Boolean — the create	whether to send invitation emails (default: false)	-		getColor() getDescription getEventById(i lld) getEventSerier Id(iCalId) getEvents(sta	iCa sBy rtTi

画面上部の検索ボックスを使用すれば、リファレンス内を検索できます。いくつかの文 字を入力すると、図 2-6-4 のようにキーワードや、ページまたはリファレンスのサジェス トが表示されますので活用しましょう。

GAS で提供されるクラスやメソッドはかなりのボリュームがありますので、すべてを網羅して覚える必要はありません。入力はコンテンツアシストを活用し、使用方法を知りたい場合は都度、本書やリファレンスを参照するとよいでしょう。

【Memo】 クラスとは何か、メソッドとは何かについては、第 6 章で解説をします。

▶図 2-6-4 リファレンス内検索

verview	SUGGESTED SEARCHES APPS SCRIPT	Table of contents
		Methods
nifaste	calendar	Detailed
ojfest structure		documentation
oogle Workspace Add-ons	「イーリートの1に作用」	createAllDayEven
endencies resource	calendar event	t(title, date)
eets resource	calendar api	createAllDayEven t(title, startDate, endDate)
oogle Workspace services	calendaris	createAllDayEven
Calendar		endDate,
Overview	DADES 1 ADDS SCRIPT	options)
CalendarApp	Advanced Celevice	createAllDayEver t(title, date, optiona)
Classes	Advanced calendar service	createAllDavEver
Calendar	Calendar manifest resource	tSeries(title,
CalendarEvent		startDate, recurrence)
CalendarEventSeries	Populate a learn vacation Calendar	createAllDayEver
EventGuest	Class Calendar	tSeries(title,
EventRecurrence	/ ページの佐補	recurrence,
RecurrenceRule	Calendar Service	options)
Enums	Class CalendarApp	createEvent(title, startTime, endTime)
Color	Class CalendarEvent	createFvent(title
EventColor		startTime,
GuestStatus	Class CalendarEventSeries	endTime,
Visibility	Class CalendarTriggerBuilder	createEventErom
Contacts		Description(desc
Data Studio	Class CalendarEventActionResponse	ription)
Document		createEventSerie s(title, startTime
Domain 🚫	REFERENCE APPS SCRIPT	endTime,
Drive		recurrence)
Forms	CalendarApp	s(title, startTime
Gmail	Calandar	end Time,
Groups		options)
Language	CalendarEventSeries ノリファレンスの作権	deleteCalendar()
Maps		getColor()
Sites	CalendarEvent	getDescription()
Slides	CalendarEventActionResponse	getEventById(iCa
Spreadsheet		lid)
vanced Google services	CalendarEventActionResponseBuilder	getEventSeriesBy Id(iCalId)
	CalendarTriggerBuilder	getEvents(startT

さて、本章では、GAS の専用エディタである「スクリプトエディタ」とプロジェクトの 管理ツールである「Apps Script ダッシュボード」の使い方についてお伝えしてきました。 これまで見てきたように、GAS の開発をサポートするさまざまな機能が提供されていま す。上手に使いこなすことで、習得のスピードも上がることでしょう。

次章から、いよいよ本格的なプログラミングへと入っていきます。GAS のベースとなる JavaScript の基本構文について学んでいくことにしましょう。

20. プロパティサービス

20.1. プロパティストア

◆プロパティストアとは

GAS では、プロジェクトやドキュメントに紐づく形でデータを格納しておくことができます。その格納しておく領域を、プロパティストアといいます。プロパティストアには、 文字列形式のキーと値のペアを格納することができ、スクリプトから読み書きできます。

たとえば、スクリプト内で使用するファイルの ID や、外部と接続するために必要な情報 などは、スクリプトのグローバル領域に記述することもできますが、その場合プロジェク ト内のすべての領域から変数名だけでアクセスできてしまいますし、コードをコピーして 再利用するときに流出してしまうという心配があります。

そのようなときに、プロパティストアを使用することで、それらのデータをコードから 分離して安全に管理できます。また、ユーザーごとに別のデータを格納することもできま すので、ユーザーによって別のスプレッドシートを参照するといったことも可能になりま す。

[Point]

プロパティストアは、データをキーと値のペアの形式で格納する領域です。

プロパティストアには、図 20-1-1 のように、プロジェクトに紐づいて格納するスクリプ トプロパティ、ユーザーごとに格納するユーザープロパティ、ドキュメントに紐づいて格 納するドキュメントプロパティという 3 種類がありますので、用途に応じて使い分けるこ とになります。

▶図 20-1-1 プロパティストアの種類

プロパテ ィストア	プロパティのアクセス	例
スクリプ トプロパ ティ	スクリプト単位で保有するスク リプトプロパティにアクセス	スクリプトで使用するフォルダやファ イルのID、DB接続のアカウント情報 など
ユーザー プロパテ ィ	スクリプトを実行するユーザー 自身のユーザープロパティにア クセス	ユーザーごとに作成したファイルIDな ど
ドキュメ ントプロ パティ	開いているドキュメントのドキ ュメントプロパティにアクセス	ドキュメントに使用されているデータ のソースURLなど

20.2 Properties サービスとPropertiesService クラス

◆ Properties サービスと PropertiesService クラス

Properties サービスはプロパティストアを取り扱うためのサービスで、Script Services に 含まれています。Properties サービスには、図 20-2-1 に挙げる 2 つのクラスが用意されい ます。

▶図 20-2-1 Properties サービスのクラス

クラス	説明
PropertiesService	Propertiesサービスのグローバルオブジェクト
Properties	 プロパティストアを操作する機能を提供する

Properties サービスを使用することで、プロパティストアへのデータの読み書きができま すが、第1章でお伝えした割り当て、制限により、1日あたりのプロパティの読み書きが できる回数や、プロパティとして持てるデータサイズが定められています。その範囲で使 用するように注意しましょう。

【Point】 プロパティの読み書きの回数や格納する容量に注意しましょう。

PropertiesService クラスでは、図 20-2-2 に示す通り、各プロパティストアをProperties オブジェクトとして取得するメンバーが提供されています。

▶図 20-2-2 PropertiesService クラスのメンバー

メンバー	戻り値	説明
getScriptProperties()	Properties	スクリプトプロパティを取得する
getUserProperties()	Properties	ユーザー自身のユーザープロパティを取得する
getDocumentProperties()	Properties	ドキュメントプロパティを取得する

スクリプトプロパティを取得する getScriptProperties メソッド、ユーザープロパティを 取得する getUserProperties メソッド、ドキュメントプロパティを取得する getDocumentProperties メソッドは、以下の書式でそれぞれのプロパティストアの内容を Properties オブジェクトとして返します。 ▶構文

PropertiesService.getScriptProperties()

PropertiesService.getUserProperties()

PropertiesService.getDocumentProperties()

実際にプロパティストアの内容を操作する際には、これにより取得した Properties オブ ジェクトに対してデータの読み書きなどの操作を行うことになります。なお、 getUserProperties メソッドでは、現在スクリプトを実行しているユーザーに紐づくユーザ ープロパティを取得します。

例として、サンプル 20-2-1 を実行してみましょう。この例では、スクリプトプロパティ を設定し、またそれを取り出してログ出力するものです。

ここで、setProperties メソッドと getProperties メソッドは、それぞれ Properties オブジェクトのキーと値のペアを設定または取得するものです。

▶サンプル 20-2-1 スクリプトプロパティの設定と取得 [sample20-02.gs]

```
function myFunction20_02_01() {
  const scriptProperties = PropertiesService.getScriptProperties();
  scriptProperties.setProperties({'犬': 'わんわん', '猫': 'にやあにやあ',
  '狸': 'ぽんぽこ'});
  console.log(scriptProperties.getProperties());
}
```

◆実行結果

{'犬': 'わんわん', '猫': 'にゃあにゃあ', '狸': 'ぽんぽこ'}

この例は、あまり意味のあるデータではありませんが、プロパティストアを使ってスク リプト内で使用する他のファイルの ID や、API 接続で使用するトークンなど、定数的に使 用するータや流出を避けたいデータなどを管理するとよいでしょう。

(Memo)

以前、スクリプトプロパティはスクリプトエディタのメニューから閲覧および編集をすることが可能でした。しかし、2020 年 12 月に新しい IDE が導入された際に、その機能は取り除かれました。したがって、新しい IDE でプロパティストアを使用する際には、最初の手順として、そのデータを設定するためのスクリプトを作成、実行することが必要になっています。

20.3. Properties クラス - プロパティストアの読み書き

◆Properties クラスとは

Properties クラスは、プロパティストアを操作する機能を提供するクラスです。図 20-3-1のようにプロパティストアのデータを読み書きする、または削除をするメンバーで構成さ れています。

▶図 20-3-1 Properties クラスのメンバー

メンバー	戻り値	説明
deleteAllProperties()	Properties	プロパティストアのすべてのプロパティを削除 する
deleteProperty(key)	Properties	プロパティストアの指定されたkeyを持つプロ パティを削除する
getKeys()	String[]	プロパティストアのすべてのキーを取得する
getProperties()	Object	プロパティストアのすべてのキーと値のペアを 取得する
getProperty(key)	String	プロパティストアの指定されたkeyに関連付け られた値を取得する
setProperties(properties[, deleteAllOthers])	Properties	プロパティストアにpropertiesで指定したすべ てのキーと値のペアを設定する (deleteAllOthersにtrueを設定すると、事前に 既存のすべてのプロパティを削除する)
setProperty(key, value)	Properties	プロパティストアに指定されたkeyと値valueの ペアを設定する

◆キーと値のセットをまとめて読み書きする

プロパティストアのデータをまとめて読み書きするには、getProperties メソッドと setProperties メソッドを使います。

書式は以下の通りです。

Propertiesオブジェクト.getProperties()

Propertiesオブジェクト.setProperties(オブジェクト[, 削除オプション])

getProperties メソッドは、指定したプロパティストアのすべてのキーと値のペアをオブ ジェクト形式で取得します。setProperties メソッドでは、オブジェクト形式で指定したキ ーと値の組み合わせすべてについて、キーが存在すれその値を書き換え、キーが存在しな ければキーと値を新たに追加します。削除オプションは省略できますが、true を設定する と、一度対象とするプロパティストアのすべてのデータを削除してからの設定となりま す。

例として、サンプル 20-2-1 の実行後の状態からスクリプトプロパティのデータの書き換 えと追加をしてみましょう。

▶サンプル 20-3-1 スクリプトプロパティの書き換えと追加 [sample20-03.gs]

```
function myFunction20_03_01() {
  const scriptProperties = PropertiesService.getScriptProperties();
  scriptProperties.setProperties({'猫': 'にゃーご', '馬': 'ひひーん'});
  const properties = scriptProperties.getProperties();
  for (const key in properties) console.log(key, properties[key]);
}
```

◆実行結果

馬 ひひーん 猫 にゃーご 狸 ぽんぽこ 犬 わんわん

ー方で、プロパティストアにセットされている、すべてのキーと値のペアを削除したい ときには、以下の deleteAllProperties メソッドを使用できます。 Propertiesオブジェクト.deleteAllProperties()

サンプル 20-3-2 を実行すると、実行ログには空のオブジェクトが出力されますので、確認してみましょう。

▶サンプル 20-3-2 スクリプトプロパティをすべて削除する [sample20-03.gs]

```
function myFunction20_03_02() {
  const scriptProperties = PropertiesService.getScriptProperties();
  scriptProperties.deleteAllProperties();
  console.log(scriptProperties.getProperties()); //{}
}
```

◆特定のキーの値を読み書きする

プロパティストアの操作でもっとも使用する機会が多いのは、特定のキーに対する値を 読み書きするというものです。取得には getProperty メソッドを使用し、設定には setProperty メソッドを使用します。

書式はそれぞれ、以下の通りです。

▶構文

Propertiesオブジェクト.getProperty(キー)

Propertiesオブジェクト.setProperty(キー, 値)

getProperty メソッドでは、指定したキーの値を取り出すことができます。キーが存在しない場合は、null が返ります。setProperty メソッドでは、指定したキーに値を設定します。キーが存在しない場合は、新たにキーと値をペアで設定をします。

使用例として、サンプル 20-3-3 をご覧ください。実行するたびに、スクリプトプロパティのキー COUNT に対応する値を 1 ずつカウントアップするものです。

▶サンプル 20-3-3 スクリプトプロパティの特定のキーの値の設定と取得 [sample20-03.gs]

```
function myFunction20_03_03() {
  const scriptProperties = PropertiesService.getScriptProperties();
  let count = Number(scriptProperties.getProperty('COUNT'));

  if (count) {
    count++;
    scriptProperties.setProperty('COUNT', count.toString());
  } else {
    scriptProperties.setProperty('COUNT', '1');
  }

  console.log(scriptProperties.getProperty('COUNT'));
}
```

実行するたびに、実行ログに出力される値が1ずつ増えていくことを確認してください。なお、プロパティストアの値は文字列ですから、この例では、取得した値をNumber 関数によって数値に変換したり、設定する値をtoStringメソッドで文字列に変換したりしていることを確認しておきましょう。

さて、プロパティストアにセットされている、特定のキーと値のペアを削除するには、 以下の deleteProperty メソッドを使用します。

▶構文

Propertiesオブジェクト.deleteProperty(キー)

例として、サンプル 20-3-3 の実行後の状態で、サンプル 20-3-4 を実行してみましょう。キーと値が削除されたことを確認できます。

```
function myFunction20_03_04() {
  const scriptProperties = PropertiesService.getScriptProperties();
  scriptProperties.deleteProperty('COUNT');
  console.log(scriptProperties.getProperties()); //{}
}
```

◆ユーザープロパティの活用

ユーザープロパティは、ユーザーごとに異なるデータを管理できます。たとえば、ユー ザープロパティにユーザーごとのスプレッドシート ID を格納するようにすることで、ユー ザーごとのスプレッドシートに対して処理をすることができるようになります。

例として、サンプル 20-3-5 を用意しました。ユーザープロパティに 「SPREAD_SHEET_ID」の値が存在していなければ、新規でスプレッドシートを作成し、そ の ID をユーザープロパティに格納するというものです。

▶サンプル 20-3-5 ユーザーごとのスプレッドシートを作成する [sample20-03.gs]

```
function myFunction20_03_05() {
   const userProperties = PropertiesService.getUserProperties();
   const spreadSheetId =
   userProperties.getProperty('SPREAD_SHEET_ID');
   if (spreadSheetId) {
     throw('すでにスプレッドシートが存在しています: ' + spreadSheetId);
     } else {
     const ssName = `スタッフ別(${Session.getActiveUser().getEmail()})`;
     const ss = SpreadsheetApp.create(ssName);
     userProperties.setProperty('SPREAD_SHEET_ID', ss.getId());
   }
}
```

ただし、前述の通り、ユーザープロパティは自らのデータにのみアクセスできます。で すから、スクリプトの動作を確認しづらい場合があります。その点、留意して使用するよ うにしましょう。 (Point)

ユーザープロパティは、自らのデータにのみアクセスできるので注意しましょう。

本章では、コードとは別の領域にデータを格納できるプロパティストアと、その操作を する Properties サービスについて紹介をしました。派手な機能ではないように思われるか も知れませんが、スクリプトの管理のしやすさ、読みやすさや安全性を高めることにつな がりますので、上手に活用をしていきましょう。

さて、次章はクラウド上ですべてが動作する GAS ならではの機能である、「イベント」 および「トリガー」について解説をしていきます。これらの機能を使用することで、スプ レッドシートやドキュメントの起動や編集、もしくは特定の時間帯といったイベントにあ わせてスクリプトを動作させることができるようになります。

GAS によるツールやアプリケーションの幅を大きく広げる機能となりますので、ぜひ使いこなせるようにしていきましょう。

23.1. ライブラリを使用する

◆ライブラリとその追加

GAS では、作成した関数を他のプロジェクトから利用できるライブラリという機能を使うことができます。他のユーザーが公開しているライブラリを使用することもできますし、自作をすることもできます。

すでにあるライブラリを活用することで、よく使う機能については自分で組み上げる必要がなくなり、GASの開発効率を大きく上げることにつながります。ここでは、導入テスト用に用意した「Hello」というライブラリを例として、その導入方法や使い方を見ていきます。

[Point]

ライブラリを使用することで、すでにある関数やクラスを再利用できます。

では、ライブラリの追加の方法を見ていきましょう。まず、ライブラリを追加したいプロジェクトを開き、「ライブラリ」の「+」アイコンをクリックします(図 23-1-1)。

▶図 23-1-1 ライブラリを追加する



図 23-1-2 のように、「ライブラリの追加」ダイアログが開きますので、以下ライブラリ 「Hello」の「スクリプト ID」を入力して「検索」をクリックします。

スクリプト ID: 1U81fRGYRWq-0yEJgtMIMMfxa-UrMSg03gOf99tzspsZXxt7ScZPjmIs2

(Memo)

ライブラリ「Hello」のスクリプト ID は、サンプルファイル [sample23_01.gs] 内にも記載して いますので、ご活用ください。

2	Apps Script	23章: サンプルスクリプト デブロイ マ 🛛 💿 🌒
i	ファイル	ライブラリの追加
<>>	23_01.gs 23_02.gs	利用可能なライブラリを ID で検索できます。詳細 スクリプト ID * 3VPW/g-0vF IgtMIMMfxa-I IrMSg03g0f99tzsps7Xxt7Sc7PimIs2
≡⊾	ライブラリ	ライブラリのプロジェクト設定で確認できるライブラリのスクリプト ID。
*	サービス	①スクリプトIDを入力 ②検索
		キャンセル 追加

▶図 23-1-2 「ライブラリの追加」ダイアログ

すると、図 23-1-3 のように、ライブラリ Hello についての検索結果が表示されます。 「バージョン」のプルダウンは、最新のバージョン(もっとも大きな数字)を選択しま す。また、「ID」はライブラリの識別子を表します。変更することもできますが、とくに 理由がない限りはデフォルトのままでよいでしょう。

▶図 23-1-3 ライブラリのバージョンを選択して追加

Ľ	Apps Script	23章:サンプルスクリプト デプロイマ ⑦ 🛞
í	ファイル	ライブラリの追加
$\langle \rangle$	23_01.gs	利用可能なライブラリを ID で検索できます。詳細
$\overline{\mathbb{O}}$	23_02.gs	1U81fRGYRWq-0yEJgtMIMMfxa-UrMSg03gOf99tzspsZXxt7Sc;
≡⊾	ライブラリ	ライブラリのプロジェクト設定で確認できるライブラリのスクリプト ID。
()	サービス	検索 ①バージョンを選択
		ライブラリ Hello を検索しました。
		1 *
		利用可能なパージョン。
		- ^{ID*} Hello 2IDを確認
		このプロジェクト内でこのライブラリを参照する際に使用します。
		キャンセル 追加 ③追加

そして「追加」をクリックすると、図 22-1-4 のようにスクリプトエディタの「ライブラ リ」に「Hello」が追加されます。



◆ライブラリを使用する

追加したライブラリ内のメンバーは、**識別子**を使用して、以下の記述で呼び出すことが できます。

▶構文

識別子.メンバー名

Hello ライブラリが追加されたことを、サンプル 23-1-1 を実行して確認してみましょう。「~さん、こんにちは!」というメッセージがログ出力されていれば、Hello ライブラリは正しく動作しているということになります。

```
function myFunction23_01_01() {
    console.log(Hello.getHello('Bob')); //Bob さん、こんにちは !
    console.log(Hello.getHello()); // 名無し さん、こんにちは !
}
```

なお、ライブラリを使用している場合、単一のプロジェクトよりも実行速度が遅くなり ますので、スピードを求められるプロジェクトの利用には推奨されていません。

さて、ライブラリを使用している場合、作成者がライブラリのバージョンアップをした としても、追加したライブラリのバージョンが自動的に最新になることはありません。

ライブラリのバージョンの確認や、バージョンアップを確認する場合は、「ライブラリ」から該当のライブラリ名をクリックして開く、「ライブラリの設定」ダイアログで行うことができます(図 23-1-5)。

新しいバージョンがあれば、今より大きい数字のバージョンが選べるようになっていま すので、そのバージョンに切り替えて「保存」をクリックします。

▶図 23-1-5 「ライブラリの設定」ダイアログ

2	Apps Script 23章	サンプルスクリプト デオロ	JY 👻 📋	?	***	9
0 ↔ © =, \$	Apps Script 23単 ファイル 23_01.gs 23_02.gs ライブラリ Hello ①ライブラリ名を クリック	 サンフルレベクリフト サ東行 Øデパッグ マパージョンを選択 (2)パージョンを選択 トello ライブラリの設定 パージョン パージョン 1 利用可能なパージョン。 1 ・ 	Xxt7ScZPjmIs2	以前のエ	ディク	920使用

(Memo)

公開されているライブラリは永続的に使用できるとは限りません。実際に、とても利用されているメ ジャーなライブラリでも開発がストップし、メンテナンスがされなくなるということも起こりえます。ライブ ラリは GitHub などで公開されているものも多いので、最近のアップデートがあるかなどを、活用 する際の判断材料にしましょう。

また、ライブラリの三点リーダーアイコンをクリックすると、「新しいタブで開く」 「削除」の2つのメニューが表示されます(図23-1-6)。

「新しいタブで開く」を選択すると、そのライブラリについてのドキュメントをブラウ ザの別のタブで開きます。「削除」を選択すると、ライブラリを削除できます。

▶図 23-1-6 ライブラリのその他のメニュー

2	Apps Script	23章: サンプルスクリプト デブロイ 🗸 🔤 🕅 🆃
i	ファイル	+ 5 c 記 ▶ 実行 D デバッグ myFunction23_01_01 ▼ 実行ログ 以前のエディタを使用
<>	23_01.gs	1 //「Hello」スクリプトID: 1U81fRGYRWq-0yEJgtMIMMfxa-UrMSg03gOf99tzspsZXxt7ScZPjmIs2
\heartsuit	23_02.gs	2 3 function myFunction23_01_01() {
≡⊾	ライブラリ	+ 4 console.log(Hello.getHello('Bob')); //Bob さん、こんにちは! (1) 二 占 川 - ガー マイコン、 S無し さん、こんにちは!
()	Hello	
	サービス	新しいタブで開く (2)新しいタブで開く

23.2. ライブラリを作成する

◆ライブラリのスクリプトを準備する

ライブラリを作成する方法を見ていきましょう。ライブラリを作成する手順は、以下の 通りとなります。

1. スタンドアロンスクリプトでプロジェクトを作成する

2. スクリプトを作成する

3. 共有設定で公開範囲を設定する

4. 版の管理で新しいバージョンを作成する

まず、新規のプロジェクトを作成します。**ライブラリを作成する場合はスタンドアロン** スクリプトである必要がありますので、コンテナバインドで作成しないようにしましょ う。

(Point)

ライブラリはスタンドアロンスクリプトで作成する必要があります。

また、プロジェクト名がデフォルトの識別子として採用されます。ですから、プロジェ クト名としては、JavaScript や GAS の予約語やグローバルオブジェクトとバッティングす ることはできませんし、日本語は避ける必要があります。今回は「MyLibrary」として進め ます。

続いて、サンプル 23-2-1 のスクリプトを作成します。

▶サンプル 23-2-1 ライブラリ MyLibrary [sample23-02.gs]

```
/** 人を表すクラス */
class Person {
    /**
    * Person オブジェクトを生成する
    * @param {string} name - 名前
    * @param {number} age - 年齢
    */
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
    /**
```

```
* あいさつ文をログ出力する
  */
 greet(){
   console.log(`Hello! I'm ${this.name}!`);
 }
}
/**
 * Person クラスのインスタンスを生成して返すファクトリ関数
 * @param {String} name - 名前
*@param {Number} age - 年齢(既定値は 0.1)
 * @return {Person} - 生成した Person オブジェクト
 */
function createPerson(name, age) {
 return new Person(name, age);
}
/**
* 税込み価格を返す関数
* @param {Number} price - 価格
* @param {Number} taxRate - 税率(既定値は 0.1)
 * @return {Number} - 税込価格
 */
function includeTax(price, taxRate = 0.1) {
 return price * (1 + taxRate);
}
```

人を表すクラス Person、Person クラスのインスタンスを作成する関数 createPerson、税 込み価格を返す関数 includeTax で構成されています。ライブラリ上のクラスについては、 他のプロジェクトから直接的に new 演算子によるインスタンス生成を行うことができませ んので、インスタンスを生成するための関数をライブラリ内に定義し、それを呼び出しま す。そのような、インスタンス生成用の関数を**ファクトリ関数**といいます。

ここで、図 23-2-1 のタグを用いて**ドキュメンテーションコメント**を使うことができま す。ドキュメンテーションコメントを記述した関数は、ライブラリを追加したプロジェク トにおいて補完対象にできます。

▶図 23-2-1 ライブラリのドキュメンテーションコメントで使用するタグ

タグ	説明	走書
@param	引数の情報を追加する	{データ型} 仮引数名 - 概要
@return	戻り値の情報を追加する	{データ型} - 概要

◆ ライブラリの公開範囲を設定する

次に、ライブラリを公開する範囲を定めるために、スクリプトの共有設定を行います。 なお、自分だけが使用するライブラリであれば、共有設定を変更する必要はありません。

まず、スクリプトエディタの「プロジェクトの共有」アイコンをクリック、続いて「ユ ーザーやグループ」ダイアログの「~へのリンクの権限を変更」または「リンクを知って いる全員に変更」をクリックします(図 23-2-2)。

MyLibrary アロイマ の 第 の 1 0 0 1 0 0 1 0

▶図 23-2-2 「ユーザーやグループと共有」ダイアログ

続いて、図 23-2-3 のとおり、公開範囲のプルダウンを開き、組織名または「リンクを知っている全員」を選択して「完了」とします。これで、公開範囲の設定ができました。

▶図 23-2-3 公開範囲を選択する

Ľ	Apps Script	MyLibrary ਤੋਂਟੀਕਾ		<u>°</u>	:: 🌘
i	ファイル	+ 5 さ □ > 実行 8 デバッグ test マ 実行ログ			
<>>		😁 ユーザーやグループと共有			
Ó	ライブラリ	まだ誰も制限付き			
≡⊾	サービス				
()		 ○ 株式会社プランノーツ ② 公開範囲を追 	選択		
		https:// リンクを知っている全員 ^{iq1Q44KdfsTj9pjd} リンクをつい	<u>-</u>		
		株式会社プランノーツ ・ 閲覧者 このリンクをNっているこのグループのメンバー全員が閲覧できます	•		
		①公開範囲のプルダウン			
		20 21 /** 22 * Personクラスのインスタンスを生成して返すファクトリ関数 23 * 24 * @param {String} name - 名前	完了		

さらに、ライブラリとして使用する場合には、「**デプロイ**」という手順を踏む必要があります。

デプロイというのは、公開して使える状態にするということを指します。

まず、スクリプトエディタの「デプロイ」ボタンから「新しいデプロイ」を選択します (図23-2-4)。

▶図 23-2-4 「新しいデプロイ」を選択

2	Apps Script	MyLibrary	①デプロイ - * ② ※ 🦃
i	ファイル	+ 5 d	■ ▶ 実行 ® デパック test ▼ 実行 新しいデプロイ 以前のエディタを使用
$\langle \rangle$	23_02.gs	1	/** 人を表すクラス * ②新しいデプロイ デプロイを管理
$\overline{\mathbb{O}}$	ライブラリ	+ 3	class Person { /** · Decouption = Chatterist Z · Difference オゴンティートを生まれます Z
₽,	サービス	+ 5	* Personオフシェントを主派する * @param {string} name - 名前 * @param {number} age - 年齢
		7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24	<pre>*/ constructor(name, age) { this.name = name; this.age = age; /** * あいさつ文をログ出力する */ greet(){ console.log(`Hello! I'm \${this.name}!`); } } /** * Personクラスのインスタンスを生成して返すファクトリ関数 * * @param {String} name - 名前 </pre>

すると、「新しいデプロイ」ダイアログが開きます(図 23-2-5)。ここでは、種類の選 択の設定アイコンから「ライブラリ」を選択してください。

▶図 23-2-5 デプロイの種類を選択

2	Apps Script	新しいデプロイ			?	***	?
i	ファイル	種類の選択	 ・ ・	?	以前の	エディク	タを使用
<>	23_02.gs		ウェブアプリ				
Ö	ライブラリ		実行可能 API				_
≡⊾	サービス						
*			ライブラリ (2) フイノフリ (2) デブロイタイブを選択してください				
			キャンセル デブロー	ſ			
		24	* @param {String} name - 名則				

続いて、ライブラリの説明を入力します(図 23-2-6)。ここで、説明を日本語で入力を するとライブラリ導入時にそのバージョンが認識されなくなってしまいますので、アルフ ァベットで入力する必要があります。注意してください。

入力が完了したら「デプロイ」をクリックします。

▶図 23-2-6 説明を入力してデプロイ

2	Apps Script	新しいデプロイ			?	***	9
i	ファイル	種類の選択	()	設定 ⑦	以前の	エディ	タを使用
<>	23_02.gs	ライブラリ		識(①説明を入力)			
Ó	ライブラリ			新しい説明文			_
≡⊾	サービス						
()				ライブラリ			
				他のユーザーとグループがこのライブラリを使用できるようにするには、このプロ ジェクトを共有します。			
				②デプロイ			

これで、新しいバージョンがデプロイされ、図 23-2-7 の画面となりますので、「完了」 をクリックします。なお、ここでデプロイ ID やライブラリの URL は不要ですので、コピー の必要はありません。

▶図 23-2-7 デプロイの完了

🎒 Apps Script		新しいデプロイ	?		
í	ファイル	デプロイを更新しました。	以前の	エディ	タを使用
$\langle \rangle$	23_02.gs				
Ö	ライブラリ	ハーション1 (2月26日16:48) デブロイ ID			
≡⊾	サービス	AKfycbzMSM5dMaI5IH7ZyLro3bbdv7wMNoq-yxSyX825hJQaL95AWmxt-QQGWA			
÷		 ロコピー 			
		ライブラリ			
		URL			
		https://script.google.com/macros/library/d/1mHWPjKjT9ISo4rSyrKq1Q44KdfsTj9pjdbYfssxq6Zvjzr7z9FcHIDIM/1 Iローロピー			
		他のユーザーとグループがこのプロジェクトをライブラリとして使用できるようにするには、このプロジェクトを共有します。			
		完了	< 1	了	
		24 * @param {String} name - 治則			

他のプロジェクトでライブラリの追加をするには、スクリプト ID が必要でした。スクリ プト ID は「プロジェクトの設定」メニューを開き、「スクリプト ID」のパートに表示され ています。「コピー」をクリックして取得しましょう(図 23-2-8)。

▶図 23-2-8 スクリプト ID のコピー

🄐 Apps Script MyLibrary 🛛 🖓 🎬 🌸						
í	プロジェクトの設定					
$\langle \rangle$						
\heartsuit	全般設定					
=, [‰]	①プロジェクトの設定 る設定です。これらの設定を変更しても既存のデプロイメン					
	✓ キャッチされなかった例外を Cloud の□グに記録する					
	✓ Chrome V8 ランタイムを有効にする					
	□ 「appsscript.json」マニフェスト ファイルをエディタで表示する					
	ID					
	ID は、Apps Script プロジェクトの一意の識別子です。					
	スクリプトID 1mHWPjKjT9lSo4rSyrKq1Q44KdfsTJ9pjdbYfssxq6Zvjzr7z9FcHlDlM ②コピー					

なお、自身が編集可能なライブラリであれば、「ライブラリの追加」ダイアログにおいて「HEAD(開発モード)」というバージョンを選択できます(図 23-2-9)。

▶図 23-2-9 開発モードでライブラリを追加する

2	Apps Script	
3) () () () () () () () () () (Apps Script ファイル 23_01.gs 23_02.gs ライブラリ Hello サービス	23 テ・++、 デー・フ・ケー・デーレー ② ② ③ ライブラリの追加 利用可能なライブラリを ID で検索できます。詳細 スクリプト ID * 1mHWPjKjT9ISo4rSyrKq1Q44KdfsTj9pjdbYfssxq6Zvjzr7z9FcH ライブラリのプロジェクト設定で確認できるライブラリのスクリプト ID。 jmIsz 検索 ライブラリ MyLibrary を検索しました。 パージョン HEAD (開発モード) 利用可能なパージョン。 ID *
		MyLibrary このプロジェクト内でこの ライブラリ を参照する際に使用します。 キャンセル 追加

このバージョンでライブラリを追加すると、ライブラリの変更が保存された際に、その 変更が即時反映されるようになります。ライブラリの開発時などに活用しましょう。 では、このスクリプト ID を用いて他のプロジェクトにライブラリ「MyLibrary」を追加し ましょう。その上で、サンプル 23-2-2 を作成して、ライブラリが使用できるか確認してみ てください。

▶サンプル 23-2-2 他のプロジェクトから自作のライブラリを使用する [sample23-02.gs]

```
function myFunction23_02_02() {
  const p = MyLibrary.createPerson('Bob', 25);
  console.log(p); //{ name: 'Bob', age: 25 }
  p.greet(); //Hello! I'm Bob!
  console.log(MyLibrary.includeTax(1000)); //1100
  console.log(MyLibrary.includeTax(1000, 0.08)); //1080
}
```

本章では、ライブラリとは何か、およびライブラリの作成の仕方について解説しました。

ライブラリを活用することで、効果的にコードの再利用を行うことができるようになります。ぜひ、お気に入りのライブラリを作成してみてください。