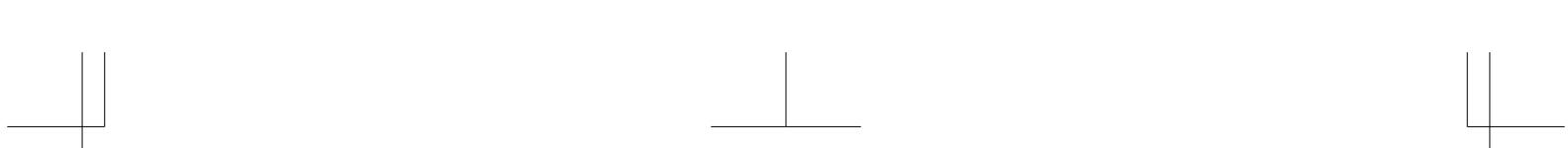


現場で使える!自動化入門

北崎 恵凡、山田 雄一、鎌田 誠、瀧川 大樹、青木 敬樹、大野 泰歩、松岡 光隆、新山 実奈子、百合宮 桜、小崎 肇、澁谷 匠、慈英 著

2023-07-31 版 発行



はじめに

お読みいただきありがとうございます。

某通信会社でインフラ設備の運用保守業務を担当し、日夜、現場に根ざした自動化・効率化に取り組み、日常の課題を解決しています。

SRE(Site Reliability Engineering)を目指して活動していますが、運用保守業務はいわゆる「コストセンター」と呼ばれ、サービスやシステムの信頼性を高める活動や付加価値を創造する活動にもあまりコストを掛けられません。

既刊「現場で使える!Google Apps Script レシピ集」、「図解と実践で現場で使えるGrafana」の執筆メンバーに加えて、本書は RPACommunity の有志が自動化のノウハウをまとめたものです。

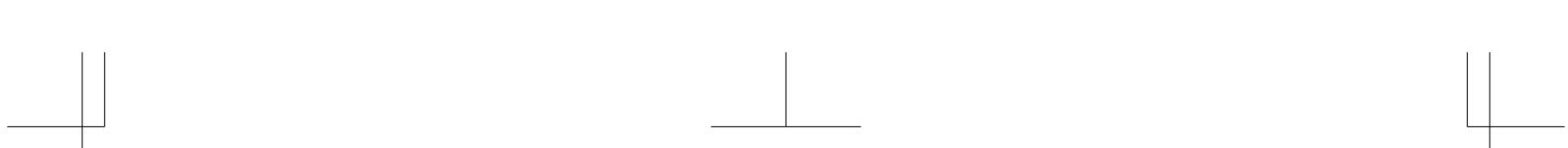
普段、現場で自動化に取り組んでいる経験が誰かのお役に立てれば幸いです。

2023年7月
北崎 恵凡

表記関係について

本書に記載されている会社名、製品名などは、一般に各社の登録商標または商標、商品名です。

会社名、製品名については、本文中では©、®、™マークなどは表示していません。



目次

はじめに	iii
表記関係について	iii
第1章 RPACommunity とは	1
第2章 RPA をいれたり自動化を検討する前に考えたい事	5
2.1 はじめに	5
2.2 世の「情報システム」にも色々ある	6
2.3 SIer によるスクラッチ開発の自社基幹システム	6
2.3.1 今のシステムを改修して隙間を埋める	6
2.3.2 RPA で隙間を埋める	7
2.3.3 RPA じゃないツールで隙間を埋める	8
2.3.4 今のシステムをそのまま諦めて使う	8
2.3.5 人を増やしてオペレーションで解決	8
2.4 パッケージソフトを使った自社基幹システム	8
2.4.1 今のシステムを改修して隙間を埋める	8
2.4.2 RPA で隙間を埋める	9
2.4.3 RPA じゃないツールで隙間を埋める	10
2.4.4 今のシステムをそのまま諦めて使う	10
2.4.5 人を増やしてオペレーションで解決	10
2.5 自前システム (SQLServer+AccessVBA とかでしっかり目の構成)	10
2.5.1 今のシステムを改修して隙間を埋める	10
2.5.2 RPA で隙間を埋める	11
2.5.3 RPA じゃないツールで隙間を埋める	11
2.5.4 今のシステムをそのまま諦めて使う	11
2.5.5 人を増やしてオペレーションで解決	11
2.6 自前システム (と呼べるかどうかわからないけど Excel+VBA ぐらいの もの) や、システムっぽいものが無いかも…?	11

目次

2.6.1	今のシステムを改修して隙間を埋める	12
2.6.2	RPA で隙間を埋める、RPA じゃないツールで隙間を埋める	12
2.6.3	今のシステムをそのまま諦めて使う	12
2.6.4	人を増やしてオペレーションで解決	12
2.7	色々と書いてきましたが…	12
第 3 章	業務断捨離のすすめ	15
3.1	今日から私は庶務担当	15
3.2	はじまり	15
3.3	何をする（した）か	16
3.4	なぜ庶務業務は属人化しやすいか	16
3.4.1	庶務業務一覧の作成	16
3.4.2	当番表	18
3.4.3	情報の公開	18
3.4.4	業務確認書	19
3.4.5	報告フォーマット	20
3.4.6	Before / After シート	21
3.4.7	事務局の立ち上げ	21
3.5	WHY から始めよ	21
3.6	断捨離（できたケース）	22
3.6.1	根本原因が既に解決していた	22
3.6.2	他では簡易に実施、または、実施していなかった	25
3.6.3	機械化・自動化	25
3.7	断捨離（できなかったケース）	29
3.8	悩み（悩んだこと）	30
3.9	気づき	30
3.9.1	良かったこと	30
3.9.2	反省点	30
3.9.3	副次的效果	31
3.10	さいごに	31
第 4 章	スマートマットをハックしてさらに便利にする	33
4.1	スマートマットとは	33
4.2	スマートマットライト	35
4.3	スマートマットライト 1 と 2 の違い	39
4.4	スマートマットライトをハックする	43

4.5	パケットキャプチャで通信の内容を確認	43
4.6	ブラウザの開発者ツールで通信の内容を確認	44
4.7	API の仕様を推察	45
4.8	GAS の実装	47
4.9	結果表示	56
4.10	さいごに	58
第 5 章	RPAによるリアルタイム処理	59
5.1	RPA のメリット	59
5.2	RPA のリアルタイム処理とは?	59
5.2.1	リアルタイム処理を知る	59
5.2.2	リアルタイム処理のメリット・デメリット	61
5.3	リアルタイム処理の実装	62
5.4	リアルタイム処理の適用事例	64
5.4.1	製造業における工程間在庫移動	64
5.4.2	製造業における完成品入力	66
5.5	リアルタイム処理の課題と解決策	68
5.5.1	エラーを素早く知るには	68
5.5.2	エラーの原因を知るには	69
5.6	リアルタイム処理の今後の展望	70
5.7	最後に	70
第 6 章	Raspberry Pi と GCP を使って、SMS の E2E 監視を実装してみた!	73
6.1	はじめに	73
6.2	E2E 監視の重要性と構築した経緯	73
6.3	監視の構成	74
6.3.1	SMS 送受信シミュレータ	74
6.3.2	監視ダッシュボード	75
6.3.3	データ連携用の Web API	76
6.3.4	予算の確保	76
6.3.5	社内のセキュリティポリシーを満たす事	76
6.4	SMS 送受信シミュレータの実装	76
6.5	SMS 送受信スクリプトの処理概要	76
6.5.1	AT コマンド投入用の USB デバイスファイルの認識	77
6.5.2	SMS の送受信	77
6.5.3	Web API で送受信結果を Post	77

目次

6.6	インターネット接続処理の概要	78
6.7	ドングルに設定必要な AT コマンド	78
6.7.1	APN の設定	78
6.7.2	APN のユーザとパスワードの設定	78
6.7.3	Packet Data Protocol(PDP) 有効化	78
6.8	WebAPI の実装	78
6.8.1	ファイアウォールで接続 IP を制限	79
6.8.2	認証キーの提供と有効期限の短時間化	79
6.8.3	入力値のバリデーション機能とプレースホルダの利用	79
6.8.4	リクエスト数とリクエストエラーの監視	80
6.9	自社内製の監視ダッシュボード側の実装	80
6.10	初期費用とランニング費用	81
6.11	初期費用について	81
6.12	ランニング費用について	81
6.12.1	GAE と Cloud SQL の費用	81
6.12.2	通信事業者の基本料 + データ使用料	82
6.12.3	SMS の送信費用	82
6.13	運用について	82
6.13.1	対処 1: 複数の監視シナリオで監視する。	83
6.13.2	対処 2: 複数の監視手段を用意する	83
6.13.3	対処 3: 複数のネットワークから API を叩く	83
6.14	まとめ	83
第 7 章	現場で使える Python 自動化入門	85
7.1	はじめに	85
7.2	頭をすっきりテスト駆動開発	85
7.2.1	はじめに	85
7.2.2	テスト駆動とは?	86
7.2.3	誰が為のテスト駆動?	87
7.3	みんなで読もう Sphinx	88
7.3.1	はじめに	88
7.3.2	Sphinx 入門	90
7.3.3	型アノテーションのすすめ	93
7.3.4	現場での活用方法	95
7.4	最後の救い log 取得	95
7.4.1	はじめに	95

7.4.2	log を設定しよう	95
7.4.3	DEBUG は DEBUG	96
7.5	さいごに	97
第 8 章	GAS と AWS を使って Web 操作をハックしてみた!	99
8.1	はじめに	99
8.2	情報収集の自動化 (GAS)	99
8.3	Web 操作の自動化 (AWS)	101
8.4	GAS から Lambda の呼び出し (GAS)	105
8.5	さいごに	105
第 9 章	フローで思い通りの結果を得るために大切なこと	107
9.1	はじめに	107
9.2	テストは、2段階に分けられる	107
9.3	Case アップロードしたはずのファイルが開かない	109
9.3.1	どんなフローか	109
9.3.2	このフローで期待される正常な動作はどういうものか	110
9.3.3	テスト機能を使って、フローの動作テストを行う	111
9.3.4	出力結果が正常な動作をするか確認する	116
9.3.5	「心当たり」をすべて確認し、問題を切り分ける	118
9.3.6	実行履歴を確認してみる	119
9.3.7	ファイルサイズを確認してみる	121
9.3.8	ファイルの作成アクションを確認してみる	123
9.3.9	ファイル コンテンツ は存在するのか?	127
9.3.10	どのアクションで「ファイルのコンテンツ」を取得するか	131
9.3.11	「添付ファイルのコンテンツを取得」アクションを理解する	132
9.3.12	フローをコピーし、「添付ファイルのコンテンツを取得」アクションを追加する	133
9.3.13	ID と ファイル識別子が取得できるか確認する	135
9.3.14	「ファイルの作成」アクションの設定し直す	137
9.3.15	フローをテストし直す	139
9.4	バグのない正しいフローを作るために	145
9.4.1	テスト項目ってどうやって洗い出すの?	145
9.4.2	テスト方法と想定結果、実際の結果を一覧表にして確認する	145
9.5	終わりに	146
筆者紹介		147

目次

北崎 恵凡 (きたざき あやちか)	147
山田 雄一 (やまだ ゆういち)	147
鎌田 誠 (かまだ まこと)	147
瀧川 大樹 (たきがわ だいき)	148
青木 敏樹 (あおき ひろき)	148
大野 泰歩 (おおの やすほ)	148
松岡 光隆 (まつおか みつたか)	148
新山 実奈子 (にいやま みなこ)	148
百合宮 桜 (ゆりみや さくら)	148
小崎 肇 (こさき はじめ)	149
瀧谷 匠 (しぶや たくみ)	149
慈英 (じえい)	149

第1章

RPACommunity とは

RPACommunity 代表 Mitz (松岡 光隆)

IT を活用した業務の自動化や IT 推進全般に興味を持つ全国の有志が集うコミュニティで、2018 年 3 月に立ち上げました。

名称には「RPA」(ロボティック・プロセス・オートメーション) が付いていますが、RPA 専用のツールやサービスだけに限らず、様々なツールやプログラムも含め IT 技術全般を対象にした学びの場を提供しています。

発足から 5 年以上経ちメンバー数も増え続け 8,000 名を超えており、今現在でもコミュニティ内では様々な手法での業務自動化トークが飛び交っています。

そんな RPACommunity 内の「自動化ネタ」を、まだ RPACommunity を知らない方々にも知っていただきたいという想いを持つ有志が集って今回の書籍作成に至りました。

RPACommunity^{*1}では 2023 年 5 月時点で 300 回以上のイベントを実施しており、その資料はこちらに掲載しております。

^{*1} <https://rpacomunity.connpass.com/presentation/>

第1章 RPACommunity とは



図 1.1: connpass の QR コード

また、2020 年以降のイベントは全て YouTube^{*2}に公開しております。

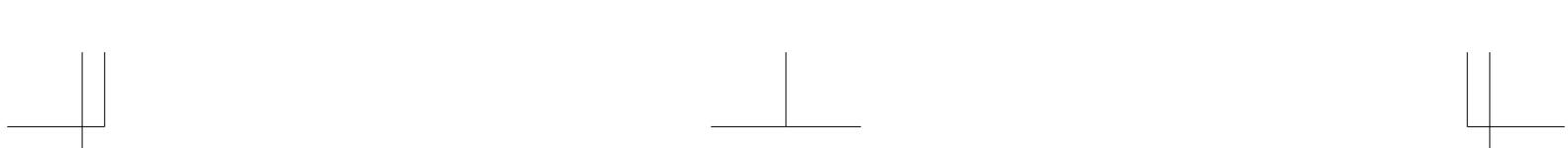
^{*2} <https://www.youtube.com/@RPACommunity>



図 1.2: YouTube の QR コード

ご興味ある方はぜひご覧ください。

本書はこれら RPACommunity での情報や登壇をベースに、RPACommunity 有志メンバーが書き上げております。



第2章

RPA をいれたり自動化を検討する 前に考えたい事

山田 雄一

2.1 はじめに

RPA を導入したり、自動化を検討したり、DX を始めたい、またそれ以前の話として現状の情報システムを改善したい……と思ったこと、ありませんか？ また、自分は乗り気でなくても、上から社命として言われてしまったり^{*1}したこと、ありませんか？

筆者としては、ここ数年（2015年、16年以降）は、RPA やノーコード、ローコードツールが一般的になった事により、単純なシステムリプレース以外に、取れる選択肢がずいぶんと増えたな……という印象があります。また既存システムにちょい足ししてみたり、既存システムのデータエントリー部分をノーコード、ローコードツールを使うようにすることで、こういった業務上の課題を解決できるかも……？ という機運が高まってきており、実際に活用例も多くなってきているように思います。

これらの RPA 系のツールは、「RPA 系のツールを使えば業務改善ができる」っぽく見える事が多いのですが、実際は打出の小槌のようにあらゆる業務改善や課題解決に効いてくれ、私達の業務を楽な方へ導いてくれるものなのでしょうか？

この章では、こういった時に何を判断基準にしたり、どういった懸念点があるのかをまとめてみました。RPA 導入以前の検討段階として、「RPA を導入したり、自動化を検討する」までに必要な検討の助けになり、ある程度^{*2}の道筋を示せれば良いなと思っています。

*1 こっちの方がもっと深刻である

*2もちろん筆者の個人的な考え方と独断と偏見によるものです

2.2 世の「情報システム」にも色々ある

世の中には色々な会社があり、色々な業務プロセスがあり、そして色々な情報システムがあります。

情報システムについては、色々なものがありますが、ターゲットを絞らないとこの手の話はついつい発散してしまうものなので、参考となりそうなケースを次の様に分けてみましょう。

- スクラッチ開発の自社基幹システム
- パッケージソフトを使った自社基幹システム
- 自前システム（SQLServer+AccessVBA とかでしっかり目の構成のもの）
- 自前システム（と呼べるかどうかわからないけど Excel+VBA ぐらいのもの）
- システムなし

これらについて、それぞれ次のような対応策を今回は考えてみましょう。

- 今のシステムを改修して隙間を埋める
- RPA で隙間を埋める
- RPA じゃないツールで隙間を埋める
- 今のシステムをそのまま諦めて使う
- 人を増やす

上で挙げたそれぞれの システム☒対応策 について、それぞれの対応策の場合の検討ポイントをそれぞれ考えていきたいと思います。

2.3 SIer によるスクラッチ開発の自社基幹システム

SIer による受諾開発（スクラッチ開発）で納品され、運用されている自社基幹システムの場合、受諾開発という契約の特性を考える必要があるケースがあります。

2.3.1 今のシステムを改修して隙間を埋める

この対応策は、開発してもらった SIer に再度追加改修をお願いする事になることが多いですが、改修時のコストがかさむ事もあります。

また、SIer の開発人員も暇ではないので、継続的に追加改修をお願いしていない場合、改めて追加改修をお願いしたとして、実際に当時作っていた人が継続して対応してくれる

2.3 SIerによるスクラッチ開発の自社基幹システム

か、品質が担保^{*3}されるか、みたいな所が悩ましくなることもあります。だいたい裏側は分かんないので……

また筆者の実体験として、継続的に改修を行っているシステムであれば、継続改修されているのでコストもそこまでかかる事もありますが、しばらく触っておらず、時間が経ってから改修を行うようなケースでは、過去のことを思い出す/既存システムの解析にもコストがかかるので、結構地味に大変だったりします。(大変なときは総じて見積もりに跳ね返ります。)

2.3.2 RPAで隙間を埋める

現行システムの改修と比べ、RPAで隙間を埋めた時の最大の利点は、「システムの内部を修正していないのに、業務オペレーションは簡単にできる(かもしれない)」点です。

RPAは良くも悪くも「画面操作をするツール」で、システムの内部を修正しませんが、これはケースによっては大きなメリットになります。

○うれしいケース1

たとえば、社内でISO認証などの外部監査がある認証を取得しており、この監査情報の取得や集約がシステムに組み込まれている場合、社内システムの改修をすると、改修箇所(時によっては監査箇所全般)の処理内容について、再度監査が必要になってしまうケースがあります。

ところが、RPAでの業務改善を行うとあら不思議! 社内システムから見ると、「画面操作をしているだけ」ですね!

これならどんな改善をしても安心!

○うれしいケース2

たとえば、最初のスクラッチ開発はSIerに頼んだが、その後の改修については他社に頼んだり、自分たちで行うケースもあるかもしれません。(プロジェクト炎上したりするとありがち)

よく受諾開発の契約書にはあるあるなのですが、システムを開発納入した会社以外が改修した場合は、納入会社はシステムの瑕疵担保責任をその後一切追わないとする契約となっているケースがあったりします。(責任を負えなくなるのでそれはまあ当たり前といえば当たり前)

こういった場合に、システムを直接改修するのではなく、RPAを使って改善を行った場合、やはり社内システムから見ると、「画面操作をしているだけ」になるのでシステム利用時の瑕疵担保責任が継続され、そういう点でも安心だったりします。

^{*3}ここで言っている品質とは単に「バグがない」というだけではなくて、ソースコードの保守性であったり、応答速度であったり、データ量であったりといった非機能要件も含みます。

第2章 RPA をいれたり自動化を検討する前に考えたい事

2.3.3 RPA じゃないツールで隙間を埋める

RPA じゃないツールで隙間を埋める というのもケースによっては非常に効きます。
基幹システムを作ったはいいが、そんなに今後長いことこれを使わないと思うので、
システム自体に改修は入れたくない、でも現状の業務は厳しいので改善をしたいとか、
RPA ではやりにくい大量データの捌きを行った上で、処理を実現する必要がある……
といった場合に、たとえば ETL ツールがハマったりする事があります。

ただ、RPA じゃないツールの場合において、データの I/O は API 経由だったり、DB
への直接アクセスが手段になりますが、パフォーマンスが劣化したり、本来の基
幹システムからのデータ I/O ではない I/O (特に書き込み処理) が発生するので、既存シ
ステムとのそういう所の統制を取りたりするのが思った以上に大変なのと、当然ながら
前述したような監査があるシステムの場合、悩ましい事になります。

2.3.4 今のシステムをそのまま諦めて使う

改修コストとそれにともなって得られるメリットが見合わない場合はこうなりがちで
す。やむを得ないです……

2.3.5 人を増やしてオペレーションで解決

人を増やして対応するはある意味最強の一手です。ハネムーン係数なども改善するか
かもしれません。やったね！ 業務が強くなるね！

※ただしボトルネックに人が増えない場合はなにも変わらず教育コストだけがかさむ
ケースも……

2.4 パッケージソフトを使った自社基幹システム

パッケージソフトの場合にはパッケージソフトならではの検討ポイントがあるのでは？
と筆者は考えています。

2.4.1 今のシステムを改修して隙間を埋める

そもそも、大規模な基幹パッケージ (例えば SAP とか) を入れていて、既に運用が回っ
ているフェーズならば、システムを改修したいと考える会社であれば、ほぼほぼアドオン
によるカスタマイズが入った状態になっているはずで、その上で現在のシステムと業務に
ギャップが発生しているので、そこに更にアドオンによるカスタマイズを付け加える形
(付け加えたいと考えている状態) になっているのではないかと考えられます。

2.4 パッケージソフトを使った自社基幹システム

ただパッケージソフトのカスタマイズというのは、一般論でいうと2割の処理にカスタマイズを入れるのであればフルスクラッチで作るのとコストが変わらないと言われているぐらいのもので、非常に開発コストがかかるものです。

また、一般的な大規模パッケージのベストプラクティスは「極力カスタマイズなしで使い、業務をパッケージに合わせる」となっている事が多く、そういった意味でも改修は悪手となりがちですが、現場要望もあって、情シスとしても押し負けてしまうことが多く、なかなかそういった理想的な形にはならない実情がありますね……

■コラム：コラム：パッケージソフトにおいて、カスタマイズがないと なにが嬉しいのか？

カスタマイズなしで使うと何が嬉しいかというと、パッケージのバージョンアップの際にカスタマイズはすべて見直しが要る事が多く、カスタマイズがなければパッケージのバージョンアップがサクサクで進むので、維持管理コストがずいぶん軽くなるのです……（カスタマイズがあればあるほど、再検証や再開発のリスクが上がる）

なお、大規模パッケージを使う時にも、会計系の箇所については原則として一切カスタマイズを入れないケースが多く、これは前述の「監査がある場合」に近い話が待っている事が多いです。

具体的には、こういったパッケージを入れる時の大きなメリットの一つとして、会計監査の省力化があげられます。

フルスクラッチで会計システムを構築した場合、大きな企業さん^{*4}では担当の会計事務所にシステムの監査をして貰う必要があり、これが経理担当の悩みのタネだったりするのですが、有名な会計パッケージであれば、それをそのまま素直に使ってさえいれば、ある程度の会計システムとしての正しさをパッケージが保証してくれるため、会計的にチェックすべき点が相当シンプルになり、会計事務所が嬉しいので、経理担当も非常に嬉しいんだそうで……

2.4.2 RPA で隙間を埋める

こういったケースでもやっぱりRPAはオイシイです。だって、画面しか触らないですからね。

また、前述の「2割の処理にカスタマイズを入れるのであればフルスクラッチで作るのとコストが変わらない」みたいな話を、パッケージソフト側ではなく、その外で持てるよ

^{*4} 上場してたりだとか…

第2章 RPA をいれたり自動化を検討する前に考えたい事

うになるので、そういう意味でも良いです。

加えてコスト面でも有利なことがあります。フルスクラッチ開発と比べても、大規模パッケージのカスタマイズができる人というのは、そのパッケージに精通したコンサルレベルのSEであることが多い、そういう人にお願いするとどうしても単価そのものが高くて、結果これが開発コストとして高くなりがちなのですが、RPAであればそういった高単価の人をアサインせざとも、業務改善が実現できるかもしれません。これは嬉しいポイントですね。

2.4.3 RPA じゃないツールで隙間を埋める

RPA じゃないツールも前項の「RPA で隙間を埋める」と同じ観点でオイシイです。ただ、内部データを直接 API で触る事になるので、投入したデータがパッケージ上でうまく処理されることを検証しておく必要があります。(筆者は実際にそういうテストをやったこともあります……笑)

2.4.4 今のシステムをそのまま諦めて使う

人生諦めも肝心です。投入コストより回収リターンのほうが少ない場合は「何もしない」が最適解だったりしますよね…(特にパッケージは改修コストがかかりがちなので、ノーカスタマイズで使って人が合わせる方が良かったりすることもままあります。)

2.4.5 人を増やしてオペレーションで解決

人こそパワー！(※前の「スクラッチ」の話と全く論点は変わらないです)

2.5 自前システム (SQLServer+AccessVBA とかでしつかり目の構成)

さて、コストをがっつりかけてSIerに基幹システムを作ってもらったりはしておらず、パッケージソフトを入れていない会社というのも意外に多いものです。(私も前職は社内SEだったのですが、このパターンでした。)

2.5.1 今のシステムを改修して隙間を埋める

自前システムだと改修しやすい面と、改修が難しくなる面があります。

改修し易いポイントは、自前で作っているのでコスト面や対応時間がコンパクトですみます。改修が難しいポイントは、そもそもそのために人数を多く割いていなかつたりする

2.6 自前システム（と呼べるかどうかわからないけど Excel+VBA ぐらいのもの）や、システムっぽいものが無いかも…？

のでマンニングが難しい事があるのと、少人数で開発をする事になるので、そんなチームにエキスパートがごろごろいるわけでもなく、結果として技術的負債が蓄積しやすい点でしょうか。（そのため、改修コストがどんどん上がっていく傾向にあります。）

2.5.2 RPA で隙間を埋める

RPA については自前システムだと悪手になるケースが多いです。（自分たちでシステムの表も裏も全部ハンドリングできている筈なので、RPA を使うより直接改修したほうが普通は望ましい筈）さて、RPA を使ったほうが良いケースはあるのでしょうか？

たとえば、自前システムを構築している基盤に無い機能を RPA で簡単に付け足せる場合などは、RPA を導入するメリットがありそうです。（正直、前職ではこのシナリオで少し導入を検討したことがあります。結局入れなかったのですが。）

2.5.3 RPA じゃないツールで隙間を埋める

外部データ連携などをする場合や、新しい機能を実現したい場合など（最近だと AI とか？）、RPA じゃない他のツールが刺さるケースもあるかもしれません。

2.5.4 今のシステムをそのまま諦めて使う

諦めも肝心です。でもどこかで重い腰をあげないといけないタイミングというのも……

2.5.5 人を増やしてオペレーションで解決

もうここまで来ると、既存のシステムを守るという観点よりは、新しいシステムや業務プロセス改善を実現するために人を増やす必要があるかもしれません。

2.6 自前システム（と呼べるかどうかわからないけど Excel+VBA ぐらいのもの）や、システムっぽいものが無いかも…？

小さい会社で簡単なシステムを作つて運用しているケースも多いでしょう。ExcelVBA などがよく活躍するケースです。あとは、「情報システム」と言われて思い浮かぶものが無いケースもあるでしょう。

でも、IT が基幹システムのようにしっかりと、連動して動くようになっていなくても、IT を使つた業務プロセスが回つているケースなども広義のシステムと言えるかもしれません。

第2章 RPA をいれたり自動化を検討する前に考えたい事

2.6.1 今のシステムを改修して隙間を埋める

簡単なシステムや小さなマクロに分割されているほど、改修というよりは1つのシステムにまとめていくほうが良いかもしれません。

2.6.2 RPA で隙間を埋める、RPA じゃないツールで隙間を埋める

ここはもうツール導入の可否みたいな話で論点がまとまっています。

そもそもの話として、RPA やその他のツールで隙間を埋めるにせよある程度の粒度のシステムになってからな気がします。その際に、ハンドオペレーションが入っている箇所をシステム化し、一気に自動化できるかもしれません。

なお、それは一般的には「DX」と言います。

改善の伸びしろがいっぱいある状態は、ある意味では良い事なのかもしれません。

2.6.3 今のシステムをそのまま諦めて使う

まあそんな判断になることもあるかもしれません。投入できるコストは有限ですし。

2.6.4 人を増やしてオペレーションで解決

これぐらいの規模感だと、「増やせるものなら増やしたい」というのが本音になりそう。

2.7 色々と書いてきましたが…

色々なケースを挙げ、RPA が効きやすそうなケース、効きにくそうなケースを様々に検討してみましたが、いかがだったでしょうか？

ちなみに情報システムの再構築における様々な論点については、「SEC BOOKS：システム再構築を成功に導くユーザガイド 第2版～ユーザとベンダで共有する再構築のリスクと対策～」という書籍があったのですが、絶版になってしまいました。ただ、PDF は無料公開されていて、誰でも読めるようになっており、ここに様々な参考となる情報が掲載されています。

<https://www.ipa.go.jp/archive/publish/secbooks20180223.html>

RPA での改善を検討する際にも、本来的なアプローチとしては

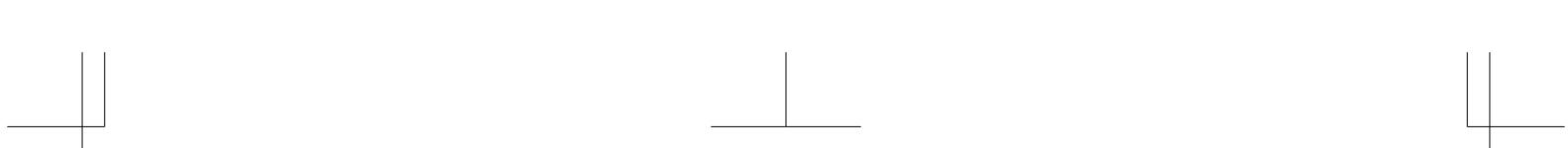
- システムリプレースや改修するほどではないね
- RPA で改善をしようね

2.7 色々と書いてきましたが…

という検討結果を経て、その後から RPA での改善が始まるのでは？ と思いますので、こういったソースにも当たってみて、一読した上で論点整理をしておくと、少し導入がスムーズになるかもしれません。⁵

この記事が最適な IT システム構築、ツール選定戦略の一助となれば幸いです。

⁵ 上長に説明するときとか、資料があると説明力が段違いだったりします



第3章

業務断捨離のすすめ

北崎 恵凡

3.1 今日から私は庶務担当

これから話す内容は架空の会社の技術部門の一組織で、本業であるエンジニアリングの傍ら、庶務担当を命ぜられ、業務断捨離を進めてきた体験談（フィクション）です。どのように考えて行動し、苦楽を経験し、業務改善してきたかの一節を共有できれば幸いです。

3.2 はじまり

少子高齢化、働き手世代の人数減少と団塊世代の引退（2025年の崖問題^{*1}）。じわじわと差し迫ってくる不安がある日、現実のものとなる。担当者が3人→1人に減り（2人は円満引退・シニア再雇用で別の道へ）、技術系管理職の宿命とは言え、すべての庶務業務が自分に回ってきた。組織変更や人事異動も重なり、業務整理をせざるを得ない状況に陥った。何から始める（た）か最初に頭の中に浮かんだのは、業務のデューデリジェンス（本来は投資を行うにあたって、投資対象となる企業や投資先の価値やリスクなどを調査すること）だった。表現として「業務整理」「業務棚卸し」でもよかったが、判断軸を設定し、価値がないものに工数（コスト）をかけてもムダだと思ったからだ。

営利企業の目的は2つしかない。

- 売上（収益）増加（P）

^{*1} 経済産業省が「DX レポート」にて提示した日本の近い将来に対する警鐘で、日本企業が DX の取り組みを十分に行わなかった場合、2025 年以降に年間で最大 12 兆円の経済損失が発生し、国際競争力を失うという課題。合わせて老朽化した IT 設備が残ると、技術的負債によりサービス競争力も低下することが懸念される。

第3章 業務断捨離のすすめ

- コスト削減 (L)

本業はインフラ設備の運用保守業務を担当し、SRE(Site Reliability Engineering)^{*2}を目指して活動していますが、運用保守業務はいわゆる「コストセンター」と呼ばれ、サービスやシステムの信頼性を高める活動や付加価値を創造する活動にもあまりコストを掛けられず、日々昼夜、自動化・効率化に勤しんでいます。庶務業務もエンジニアリングの一種と捉えて、同じように取り組めないか、と考えはじめました。

3.3 何をする(した)か

- 庶務業務一覧の作成
- 当番表の作成+ロギング・数値化
- 報告フォーマット(1枚)の作成
- Before / After シートの作成
- セルフサービス化
- コミュニケーションコストを下げる(情報の公開、チャットボットの活用)
- 機械化・自動化
- 事務局の立ち上げ

3.4 なぜ庶務業務は属人化しやすいか

本業でないことは誰も興味が無いし、担当者に丸投げ・任せきりになり、属人化しやすいです。担当者が突然いなくなると困る潜在的な問題を抱えています。業務知識を公開(属人化→形式知化)し、業務理解に努める仕組みが重要です。

3.4.1 庶務業務一覧の作成

庶務業務一覧を作成しました。

^{*2} Google 社が提唱したサービス・システムの信頼性、安定性、拡張性、効率性を高めるためのエンジニアリングの一種で、システムのエラーを防止するために、自動化、モニタリング、テスト、リカバリなどのプロセスを導入することにより、人為的なミスを減らし、高いシステム信頼性を実現することを目指した方法論。システムの性能改善を目的としたメトリクスやデータ分析に基づく意思決定も重要な要素となっている

3.4 なぜ庶務業務は属人化しやすいか

No.	業務名称	業務内容	責任者	担当者	業務確認書	更新日	工数(h/月)
1	業務A	- 項目A	責任者A	担当者A 担当者a	資料A	2023年4月30日	12
2	業務B	- 項目B - bはbbをXXまでに行う	責任者B	担当者b	-	2022年1月15日	2
3	業務C						
4							
5							
6							

図 3.1: 庶務業務一覧

- 採番
 - 業務に詳しい担当者間であれば「XX の件」で通じるかもしれません、業務内容を知らなければ説明や指示にコミュニケーションコストが発生するので、庶務業務の「No.3」の件、というようにミニマムコストで認識の齟齬が発生しないコミュニケーションが大事です。

- 業務名称
 - 誰にでも(業務を知らない人でも)内容をある程度把握できる分かりやすい名称にします。

- 業務内容
 - 軽微な内容であればここに記載します。量が多く内容が煩雑な場合は業務確認書を作成します。

- 責任者
 - 責任者不在は担当者が困ります。相談相手も報告先もなく、業務が属人化・形骸化しやすい原因を作ります。

- 担当者

- 業務確認書

- 工数(h/月)

- 定量的に数値で判断できるのは重要です。工数=コストです。業務内容に対して適切なコストかどうか(サンクコストも含めて)判断します。

カテゴリ分け、優先順位づけ(重要度と緊急性の2軸マトリクスで行うことが多いですが、困り度も考慮に含めて)行います。

第3章 業務断捨離のすすめ

3.4.2 当番表

リマインダー、チェックリスト、ロギングの機能を持ち、自動的に数値化まで行います。SREではトイル (Toil)^{*3}と呼ばれる種類の業務で、1人あたりのトイルに割かれる時間を50%未満にすることが推奨されています。

2023年5月8日	当番	担当A		
作業種別	業務	頻度	内容	実施記録
アテンド				未
確認・収集				済

図 3.2: 当番表

3.4.3 情報の公開

密室の議論は内容が不透明で属人化を生みやすいので、基本的に情報は公開するようになります。(ただし、セキュリティ上の情報機密区分に応じて内容と公開範囲は適切に設定します) 情報が公開されれば情報の方向性が PUSH 型から PULL 型になり、コミュニケーションコストが下がります。

^{*3} 時間 (Time)、オペレーション (Operation)、インシデント (Incident)、負荷 (Load) の頭字語。同じことを繰り返し手動で実行する、スケールしない作業を指します。

3.4 なぜ庶務業務は属人化しやすいか

情報の方向(PUSH型からPULL型へ)

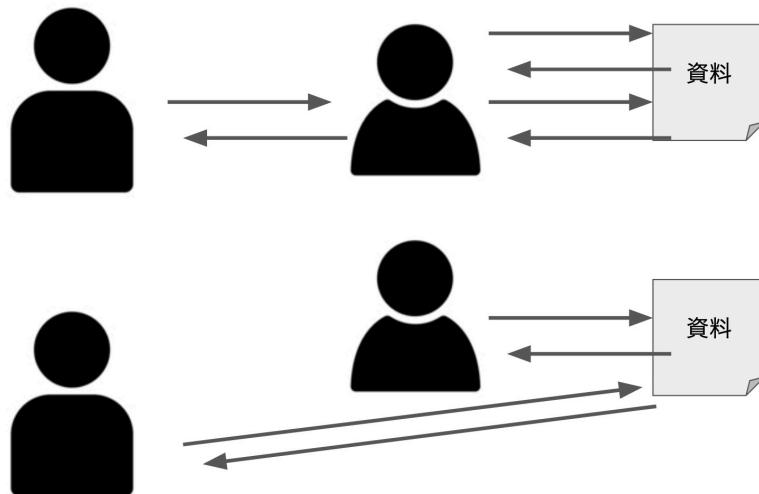


図 3.3: 情報の方向

3.4.4 業務確認書

異なる業務を同じフォーマットで見られるようにすることが重要です。

第3章 業務断捨離のすすめ

情報のフォーマット化(形式化・体系化)



図 3.4: 業務確認書

3.4.5 報告フォーマット

庶務業務は煩雑で細々とした内容が多く、情報量が多くなりがちです。整理前と整理後を1枚で見えるようにすることが重要です。

2023年4月30日時点	整理後	整理前
変更あり		
(対応必要)	チームA No.AA, BB, CC	No.MM, NN, OO, PP
(対応不要)	チームB No.DD, EE, FF	
変更なし	チームC No.XX, YY, ZZ	

図 3.5: 報告フォーマット

3.5 WHY から始めよ

3.4.6 Before / After シート

これまでの習慣や慣れた行動を変える、変えさせるのは大変です。なかなか理解は得られなくても、シンクコストを下げるために、分かりやすく伝える工夫をします。基本は「人を仲介・介在させない=セルフサービス化」、「コミュニケーションコストは最小限に」です。

Before / After

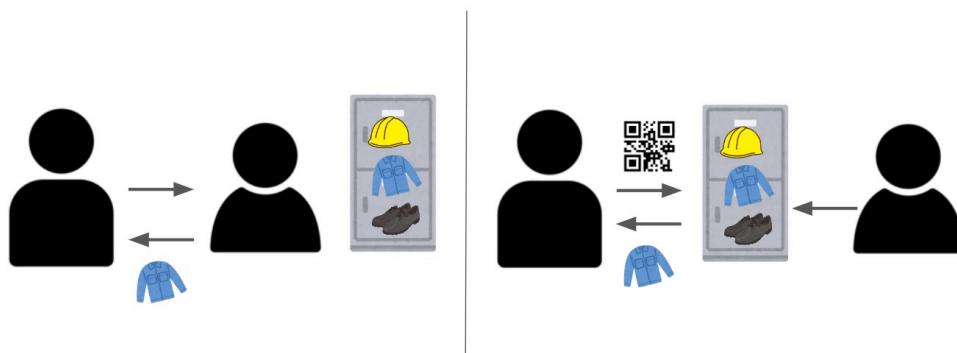


図 3.6: Before / After

3.4.7 事務局の立ち上げ

業務を前へ進めるためには、利害関係が対立した場合に備えて、取り敢えず、落とし所、最後の砦、困った時の駆け込み寺を用意することが重要です。

3.5 WHY から始めよ

ゴールデンサークル理論^{*4}を利用して業務を査定していきました。「何の業務をやっているのか」「なぜ、その業務をやっているのか」「その業務のやり方はベストなのか」業務を担当している方への敬意と感謝を忘れずに、問い合わせ続けることが重要です。

^{*4} 経営者やマーケターによって使用される、より深い目的や価値観を明確にするためのフレームワーク。

なぜ、その業務が必要なのか

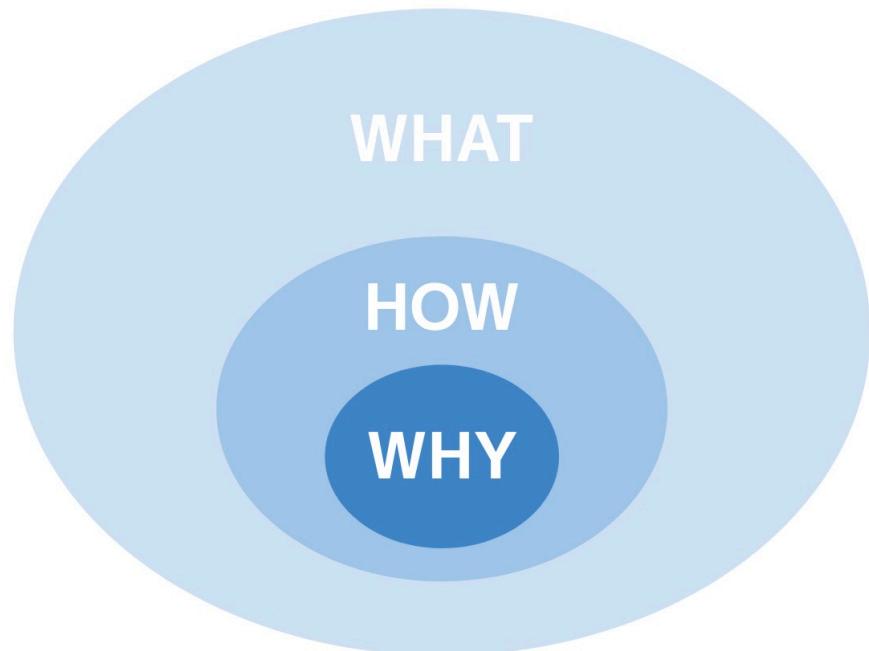


図 3.7: ゴールデンサークル

3.6 断捨離(できたケース)

3.6.1 根本原因が既に解決していた

時間経過と共にルーティン化・マンネリ化・形骸化していた。

3.6 断捨離 (できたケース)

横展開・横串 (はじまり)

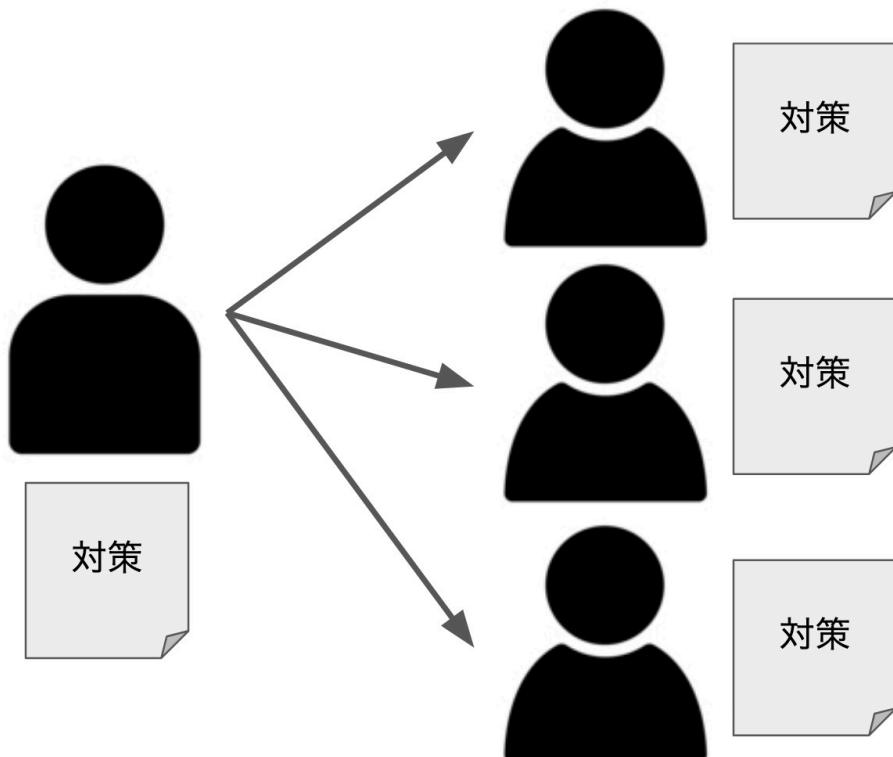


図 3.8: はじまり

横展開・横串(その後)

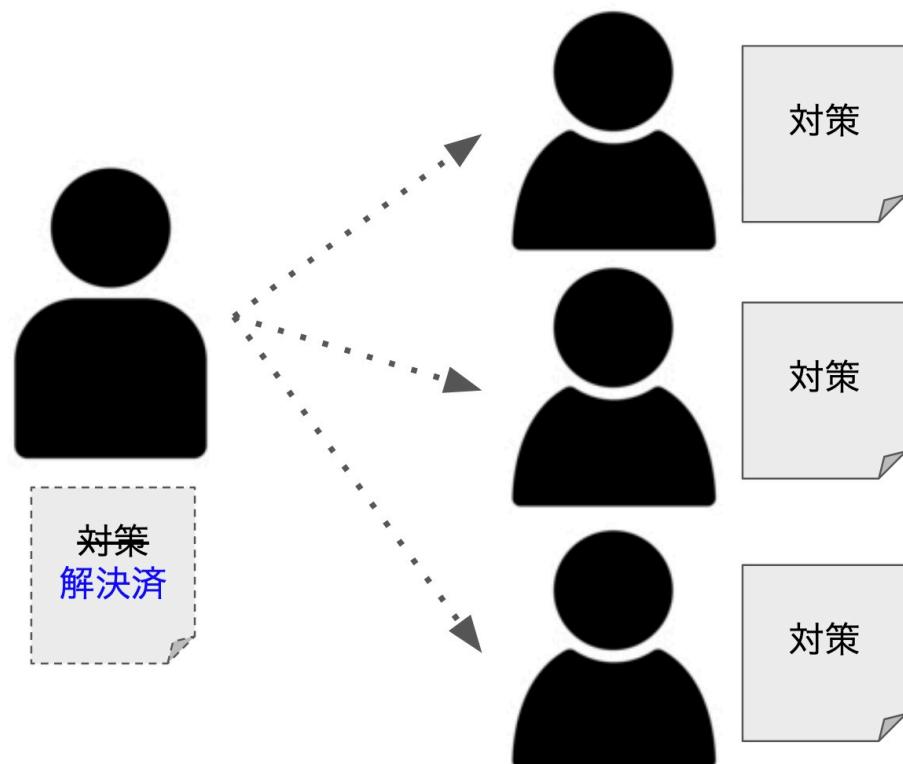


図3.9: その後

3.6 断捨離 (できたケース)

3.6.2 他では簡易に実施、または、実施していなかった

井の中の蛙

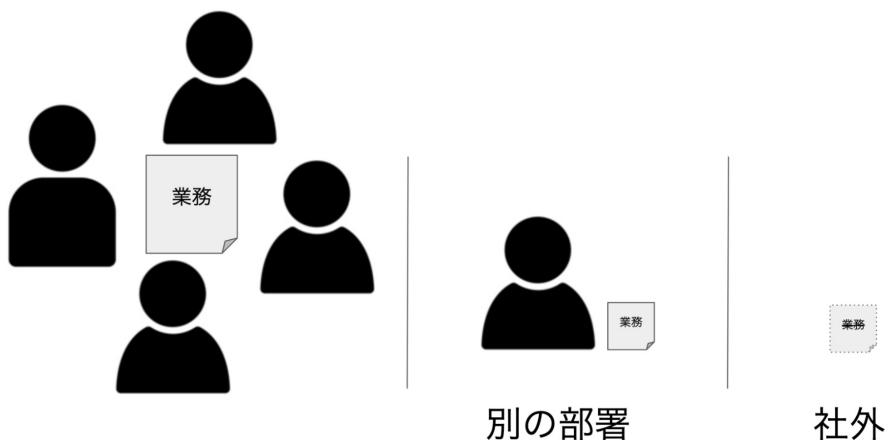


図 3.10: 井の中の蛙

3.6.3 機械化・自動化

巡回業務を無くすことができました。

- ・在庫確認 (スマートマットの活用) . . . 本書後述

第3章 業務断捨離のすすめ

マット1

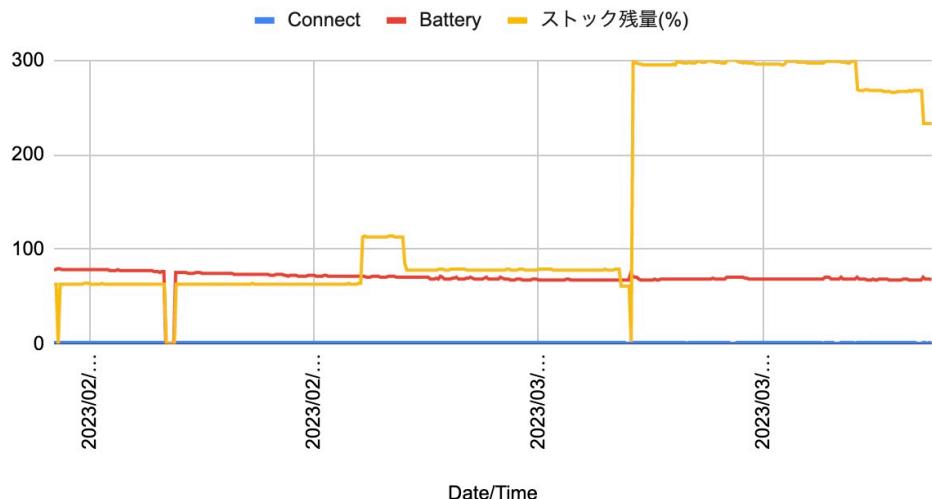


図 3.11: スマートマットライト 1

マット2

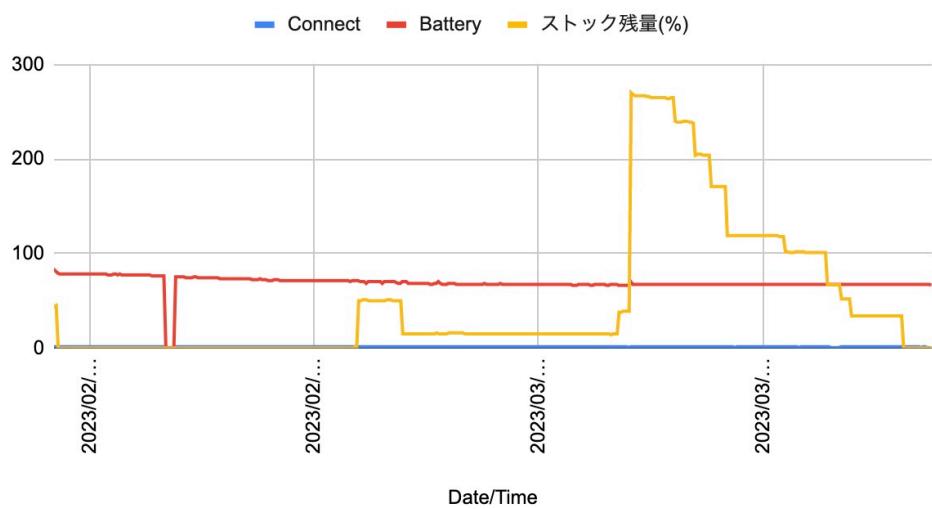


図 3.12: スマートマットライト 2

- ・滞在人数の可視化 (スマート AI カメラの活用)

3.6 断捨離(できたケース)

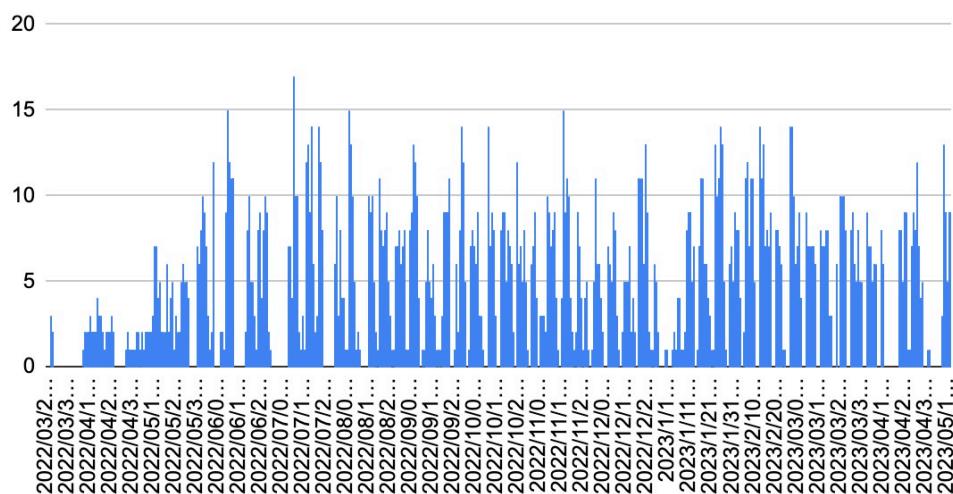


図 3.13: スマート AI カメラ 1

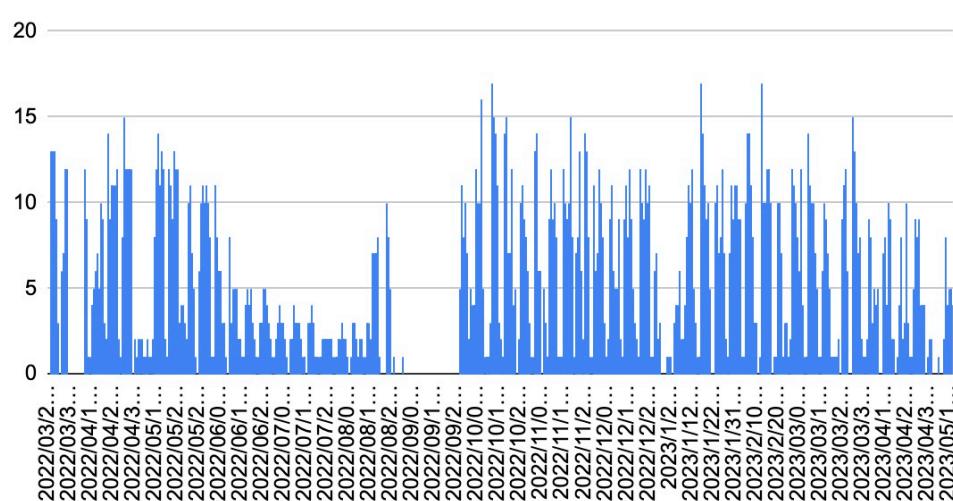


図 3.14: スマート AI カメラ 2

- ・職場環境の快適化 (IoT デバイスの活用)

第3章 業務断捨離のすすめ

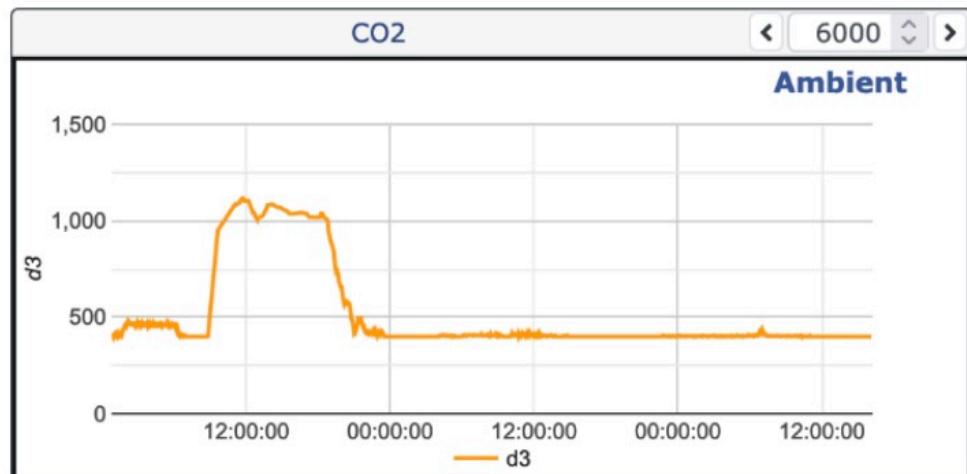


図 3.15: CO2 濃度

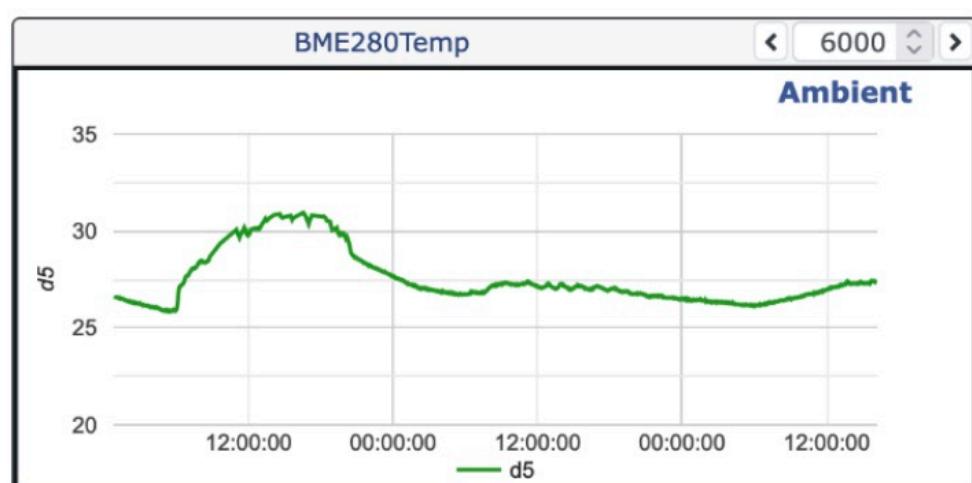


図 3.16: 気温

3.7 断捨離(できなかったケース)

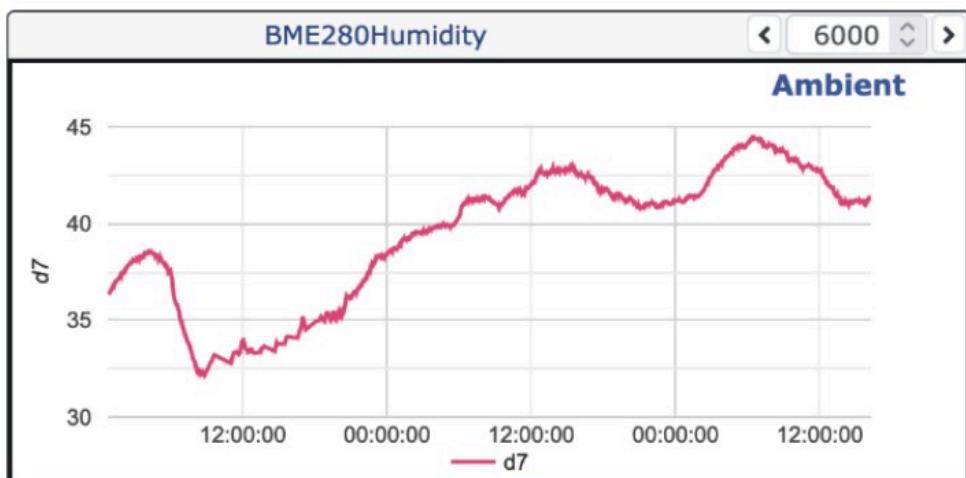


図 3.17: 湿度

3.7 断捨離(できなかったケース)

- 現場業務
 - 物理的に人を動かす必要がある場合には効率化に限界がある。
 - * (郵便物、宅配便、請求書、訪問者対応、など)
 - 頻度を減らす、まとめて対応する、などの対応方法になる。
- 法的規制
 - XX 法、施行規則、関連細則、安全衛生管理、消防法など、ルールを守る必要がある
 - * 業務は勝手に止められない。頻度・回数など。
 - 用語の解釈にも注意が必要。
 - * 「巡視」・「人が見て回ること」。ロボット化できない。
 - 政府によるアナログ規制撤廃⁵には期待しているが、対象・候補があまりなかった。
- 事務局の廃止(解散)

⁵ デジタル原則を踏まえたアナログ規制の見直し (https://www.digital.go.jp/assets/contents/node/basic_page/field_ref_resources/c43e8643-e807-41f3-b929-94fb7054377e/1420dca1/20221221_meeting_administrative_research_outline_08.pdf)

第3章 業務断捨離のすすめ

3.8 悩み(悩んだこと)

- GAS の引き継ぎ (技術スキルが必要)
 - 背伸びをせず、自分たちの身の丈にあったスキルレベルで実装が求められます。
 - * 高いスキルを求めるに、後任者がメンテナンスできなくなり、技術的負債を生みやすくなります。
- 心構え
 - そもそも、その業務を無くせないか
 - 面倒なこと、苦痛なこと(つらみ)を無くす、緩和できないか
 - 自分の仕事が無くなる不安がある
 - * (別の仕事の割り当てを先に示す必要がある)

3.9 気づき

3.9.1 良かったこと

- 一緒に活動する仲間ができた (いろいろな人と知り合えた)
- チームワークが良くなった (団結力が向上した)
- 相談相手ができた (メンタル面の支えができた)
- 業務の中身を知ることができた
- 問題・課題を具体的に把握できた
- 集合知を生かすことができた (集合知の力を発揮できた)
- 上司/部下それぞれの視点・視野・観点を理解するようになった
 - (客観的・中立的に物事を見られるようになった)
- 事務/技術スキルが身についた
 - スプレッドシート、フォーム、GAS、など
- 技術部門なのでアナログ業務(巡回)よりデジタル(コードベース)の方が
 - 理解しやすい、引き継ぎしやすい
- 技術で解決する欲求が高まった (活動のモチベーションを維持できた)
 - 滞在人数可視化、職場環境の IoT 化、計測(スマートマットもその一つ)

3.9.2 反省点

- ジョブディスクリプション(職務定義書)までは踏み込めなかった
- メンバーシップ型→ジョブ型雇用の夜明けが見られなかった

3.10 さいごに

- (差し迫った状況ではなく) 職場メンバーと(ふりかえりとして)座談会をやりたかった

3.9.3 副次的効果

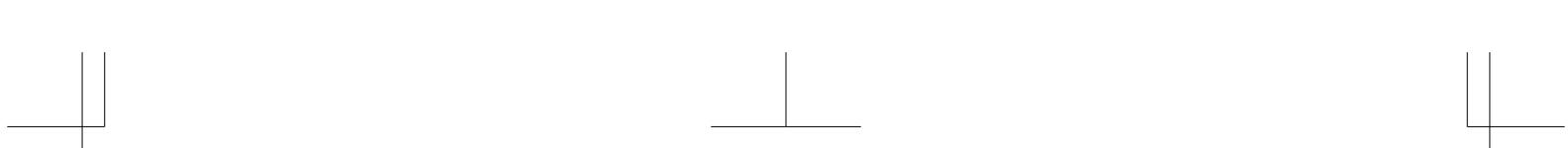
- ES サーベイ^{*6}の向上(責任者・担当者の明確化、ひとり IT 用務員の相談相手ができる)
- 心理的安全性の確保(庶務業務の一覧化、見える化(誰でも見られる場所へ公開)、数値による定量化・客觀化、負担の平準化(得手・不得手、適材適所の場合を除く))
- 担当者のローテーション(サイクル、任期、計画性が生まれた(我慢しよう、先が見えない不安から開放))

3.10 さいごに

二年間に渡る庶務業務整理がひと段落し、成功・失敗いろいろとありましたが、本書の執筆を機会にふりかえることができて良かったです。私が経験したベストプラクティスが、少しでもみなさまの「見える景色を変える」^{*7}お役に立てれば幸いです。

^{*6} ES(Employee Satisfaction) 従業員満足度調査。従業員エクスペリエンス、従業員エンゲージメントと表現されることもある

^{*7} 敬愛する沢渡あまね氏の著書から表現を借用



第4章

スマートマットをハックしてさらに便利にする

北崎 恵凡

4.1 スマートマットとは

あらかじめ登録した商品の重さを定期的に計測し、指定した条件になると自動で発注するスマートデバイスです。製品としてスマートマットライト^{*1}とスマートマットクラウド^{*2}の2種類があります。スマートマットライトはAmazonの日用品(水やペーパータオルなど、約3,000の対象商品)の自動注文に特化しており、管理画面が用意されています。マットの価格^{*3}は安価(セール時～通常:1,980～2,980円)で、マット10枚まで利用可能です。スマートマットクラウドは法人向けサービスで、棚卸自動化、入出庫管理、自動注文など、フルスペック・フルサービスの在庫管理・発注プラットフォームです。違いの詳細は製品のヘルプセンター^{*4}に掲載されています。

*1 <https://service.lite.smartmat.io/>

*2 <https://www.smartmat.io/>

*3 <https://www.amazon.co.jp/dp/B08G8B2928>

*4 <https://smartshopping.my.site.com/help/s/article/000001605>

第4章 スマートマットをハックしてさらに便利にする



図 4.1: 戸棚のスマートマットライト

4.2 スマートマットライト



図 4.2: ペーパータオルの在庫

4.2 スマートマットライト

スマートマットライトはマット本体を購入すれば月額利用料は発生しません（買い切り）。マットは単三電池 4 本で動作し、Wi-Fi へ接続します。定期的に（デフォルトでは 1 日 4 回）重さを測定し、AWS へデータが送信されます。（通信をキャプチャーして確認）データは専用の管理画面^{*5}から、対象のマットを選択→詳細情報→消費レポートへ移動して確認することができます。

^{*5} <https://lite.smartmat.io/>

第4章 スマートマットをハックしてさらに便利にする



図 4.3: スマートマットライトの管理画面

4.2 スマートマットライト

詳細情報

商品情報



Ar ハンドタオル 200枚入 5個パック >

100%時の商品重量 2.07kg >

注文設定

注文方法 買いどき通知

注文タイミング 20%以下 >

重複注文防止機能 オフ

?

重複注文防止機能とは？

残量情報

先週は 67.1%
昨日は 33.3% 商品を消費しました

商品残量 216% >

消費ベース/日 8.1% >

最終計測日時 2023/03/29 12:49

計測頻度 1日8回 >

?

消費レポートを詳しく見る >

図 4.4: 詳細情報

消費レポート

先週は**67.1%**、昨日は**33.3%**商品を使いました

注文回数

2回

消費ベース/日

8.1 %

使い切るまでの
平均日数

12.3 日

現在の商品残量

?

計測誤差がある場合

216 %

残量グラフ

1週間

1ヶ月

1年



図 4.5: 消費レポート

4.3 スマートマットライト 1 と 2 の違い

4.3 スマートマットライト 1 と 2 の違い

結論から書くと、機能上の違いはほとんどありません。スマートマットライトの裏側のシール(図 4 の矢印)を剥がすとネジが 1 本現れ、プラスドライバーで外すことができます。側面はガラス板とプラスチックの境界(図 5 の矢印)にマイナスドライバーを差し込んで開けることができます。スマートマットライト 1 は ESP-WROOM-02(ESP8266 チップ) 基板の Wi-Fi アンテナを使用していますが、スマートマットライト 2 は Wi-Fi アンテナ(図 6 の矢印)が外しとなり、ネットワークの接続性が向上しています。



図 4.6: スマートマットライトの裏側

第4章 スマートマットをハックしてさらに便利にする

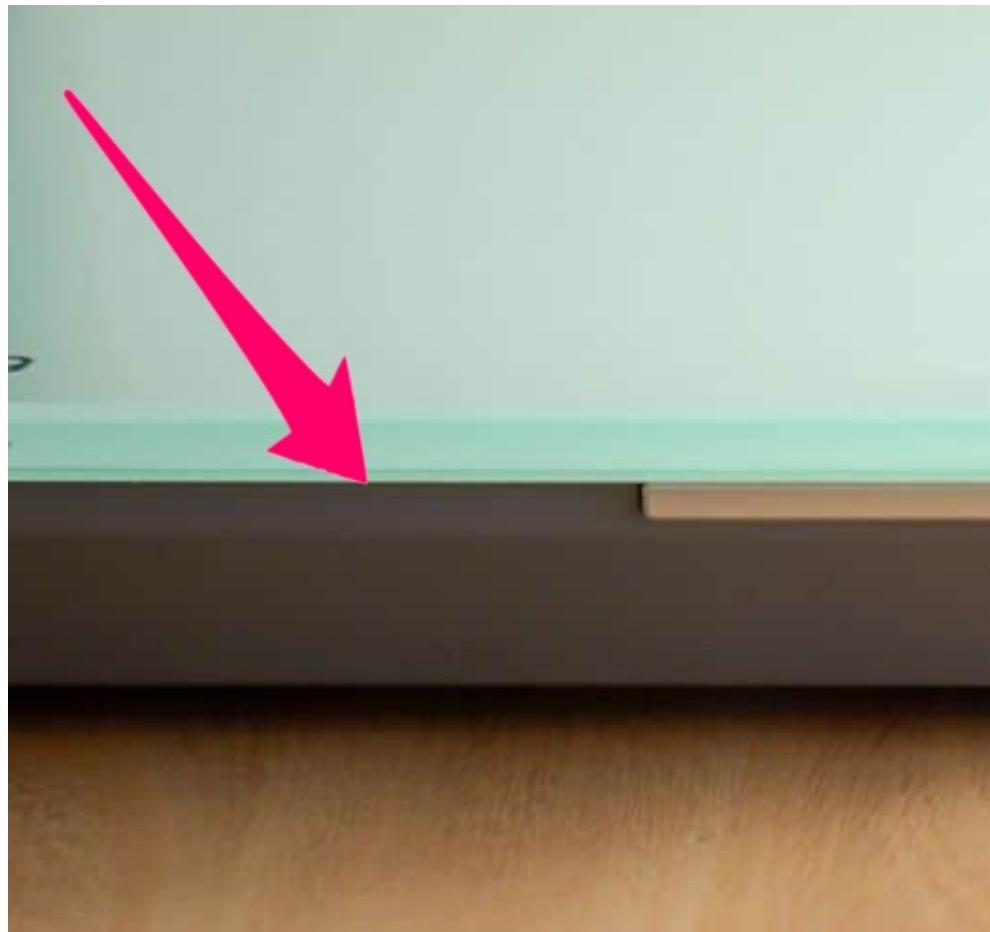


図 4.7: スマートマットライトの側面

4.3 スマートマットライト 1 と 2 の違い



図 4.8: 分解した中身 (バージョン 1)

第4章 スマートマットをハックしてさらに便利にする

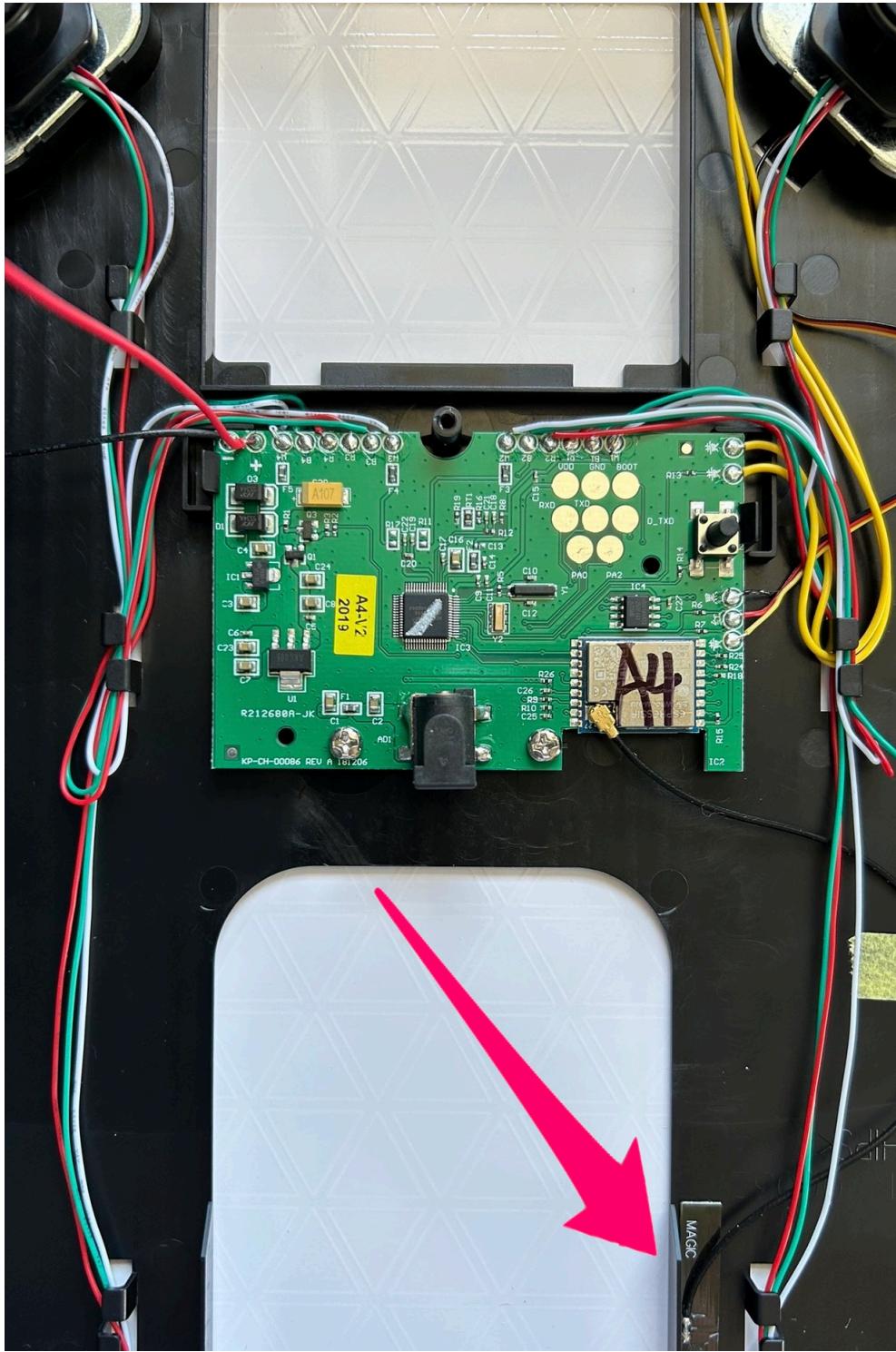


図 4.9: 分解した中身 (バージョン 2)

4.4 スマートマットライトをハックする

4.4 スマートマットライトをハックする

Amazon アカウントと連携して自動注文してくれたり、E メールや LINE(ID 連携が必要) で通知されるので非常に便利ですが、注文先を変更したり、消費データだけを利用(分析)したり、他の機能と連携させたい場合があります。これらのニーズを実現するためスマートマットクラウドは API 連携^{*6}が可能ですが、スマートマットライトは API が提供されていません。そこでこの記事では、Google Spreadsheet と GAS(Google Apps Script) を使用して管理画面の消費レポートからデータを取得し、他の機能と連携させる方法を説明します。

4.5 パケットキャプチャで通信の内容を確認

まず、macOS のインターネット共有の機能を利用して、スマートマットライトの通信内容を Wireshark^{*7}でパケットキャプチャします。



図 4.10: 接続構成

すると、以下の処理をしていることが確認できます。

1. DNS でネームを検索
2. HTTP POST で測定データを送信
3. HTTP GET でデータを取得

^{*6} <https://smartshopping.my.site.com/help/s/article/000001143>

^{*7} <https://www.wireshark.org/>

第4章 スマートマットをハックしてさらに便利にする

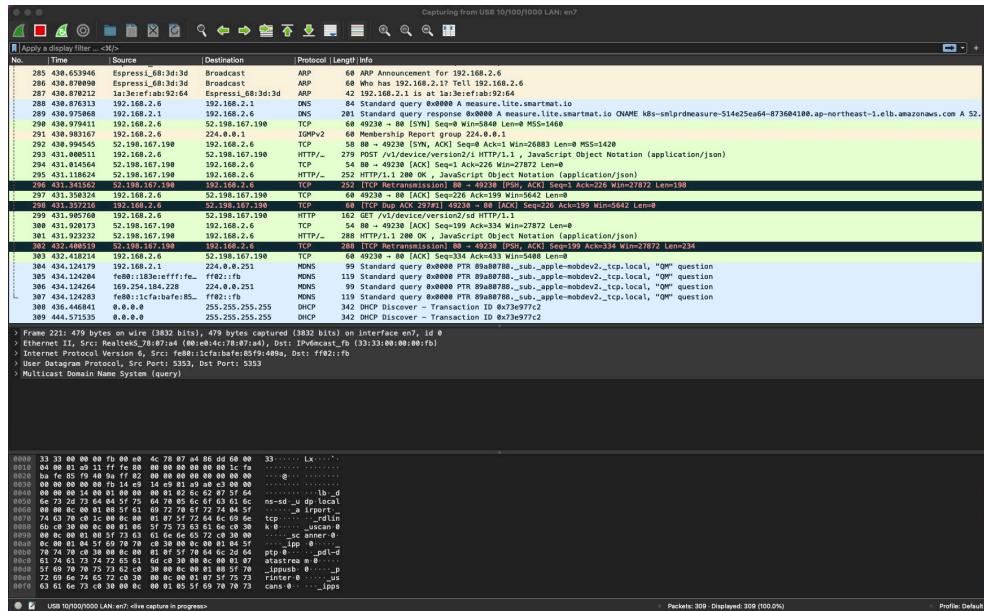


図 4.11: パケットキャプチャ

4.6 ブラウザの開発者ツールで通信の内容を確認

ブラウザの開発者ツールを使用して、スマートマットライトの管理画面にログインします。すると、以下の処理をしていることが確認できます。

1. サインインの URL(/api/bff/v1/signin) で ID(email) とパスワード (password) を送信
2. クッキー (cookie) でセッション ID(sid) を受信

4.7 API の仕様を推察

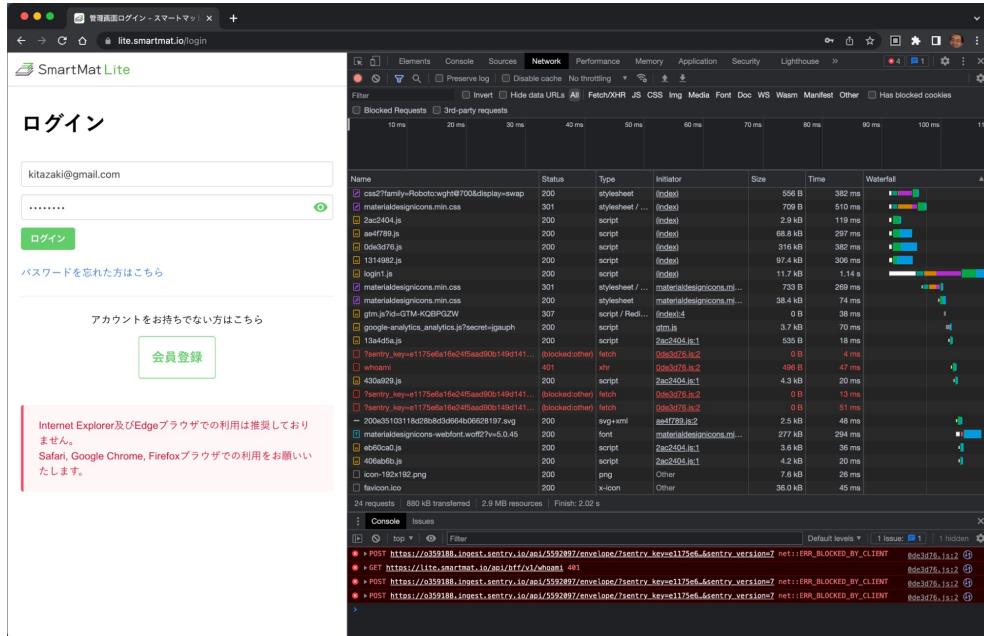


図 4.12: ブラウザの開発者ツール

4.7 API の仕様を推察

管理画面にログインした後、スマートマットのデータを確認します。詳細情報を確認すると、スマートマット毎に URL に管理番号 (subscriptionId) が付与されています。

第4章 スマートマットをハックしてさらに便利にする



図 4.13: スマートマットの詳細情報

HTTP GET でデータ取得用の URL(/api/bff/v1/subscription/search_detail) にアクセスし、以下の JSON データを取得しています。

```
{  
    "deviceSerialNumber": "WXXXXXXXXXXXXX",  
    "isFirstConnected": false,  
    "isConnected": true,  
    "battery": 66,  
    "remainingPercent": 0,  
    "triggerRemainingPercent": 20,  
    "measuredAt": "2023-04-08T06:06:03+09:00",  
    "current": 0,  
    "frequency": 8,  
    "subscriptionId": 8594,  
    "title": "Ar ハンドタオル 200枚入 5個パック",  
    "full": 2070.21,  
    "productUrl": "https://www.amazon.co.jp/dp/B00KOZ0ZFU",  
    "imageUrl": "https://m.media-amazon.com/images/I/414WdyYdVuL._SL500_.jpg",  
    "outputType": 4,  
    "orderable": false,
```

4.8 GAS の実装

```
"dailyAverageConsumption": 6.5,  
"dailyConsumption": 0,  
"weeklyConsumption": 0,  
"totalOrderCount": 4  
}
```

これで、スマートマットクラウドの API から取得できるデータとフォーマットが一部共通であることを確認できました。

表 4.1: データフォーマット

戻り値	説明	具体例
deviceSerialNumber	シリアル番号	W42190800XXX
isFirstConnected	(不明)	
isConnected	Wi-Fi 接続状況	true
battery	電池残量	73
remainingPercent	最新計測時刻の残量 % 表示の場合のみ	57
triggerRemainingPercent	閾値 (%表示の場合)	20
measuredAt	最新計測時刻	
current	最新計測値 (単位はグラム)	11330
frequency	計測頻度 (1 日 n 回)	1
subscriptionId	(不明)	
title	商品名	お米
full	満タン重量 % 表示の場合のみ	20000
productUrl	(不明)	
imageUrl	商品画像 URL	
outputType	発注通知方法	メール通知
orderable	(不明)	
dailyAverageConsumption	(不明)	
dailyConsumption	(不明)	
weeklyConsumption	(不明)	
totalOrderCount	(不明)	

4.8 GAS の実装

通信の確認と API の解析結果をもとに、GAS(Google Apps Script) を作成します。作成するスクリプトは大きく 2 つの処理から成ります。

1. スマートマットライトの管理画面へログインし、スマートマットの測定データを取

第4章 スマートマットをハックしてさらに便利にする

得する

2. 取得したデータをスプレッドシートに記録する

GAS の中の変数に必要な情報を設定します。

- ・「email」 → 管理画面のログイン ID
- ・「password」 → 管理画面のパスワード
- ・「XXXX」 → スマートマットの管理番号
- ・「sheet_id」 → スpreadSheet ID
- ・「sheet_name」 → スpreadSheet のシート名

GitHub にサンプルコード^{*8}を載せましたので参考にしてみてください。

```
function fetchJSONData() {  
  
    const postdata = {  
        'email': '', // ID  
        'password': '' // Password  
    }  
  
    const loginUrl = 'https://lite.smartmat.io/api/bff/v1/signin';  
    // ログインURL  
    const dataUrl = 'https://lite.smartmat.io/api/bff/v1/subscription/search_detail?subscriptionId=XXXX'; // データURL  
    const sheet_id = ''; // SpreadSheetID  
    const sheet_name = '' // SpreadSheetName  
  
    const options = {  
        'method': 'post',  
        'Content-Type': 'application/json',  
        'payload': JSON.stringify(postdata),  
        'followRedirects': false  
    }  
  
    // ログインページにPOSTリクエストを送信して、Cookieを取得する  
    const response = UrlFetchApp.fetch(loginUrl, options);  
    const headers = response.getHeaders();  
    const cookie = headers['Set-Cookie'];  
  
    // Cookieを使用して、JSONデータを取得する  
    const jsonData = UrlFetchApp.fetch(dataUrl, {  
        'headers': {  
            'Cookie': cookie  
        }  
    })
```

^{*8} <https://github.com/kitazaki/smartmat/blob/main/smartmat.gs>

4.8 GAS の実装

```
}).getContentText();

// JSONデータを解析する
const parsedData = JSON.parse(jsonData);

// スマートマットがWi-Fiに接続されている場合: true → 1、切断されて→
//いる場合: false → 0
let connect = 0;
if (parsedData['isConnected']) {
    connect = 1;
}

// JSONデータをSpreadSheetに出力する
const MySheet = SpreadsheetApp.openById(sheet_id);
MySheet.getSheetByName(sheet_name).appendRow([
    [new Date(), connect, parsedData['battery'], parsedData[→
    'remainingPercent']]
]);
}
```

GAS を保存したら「実行」を押します。

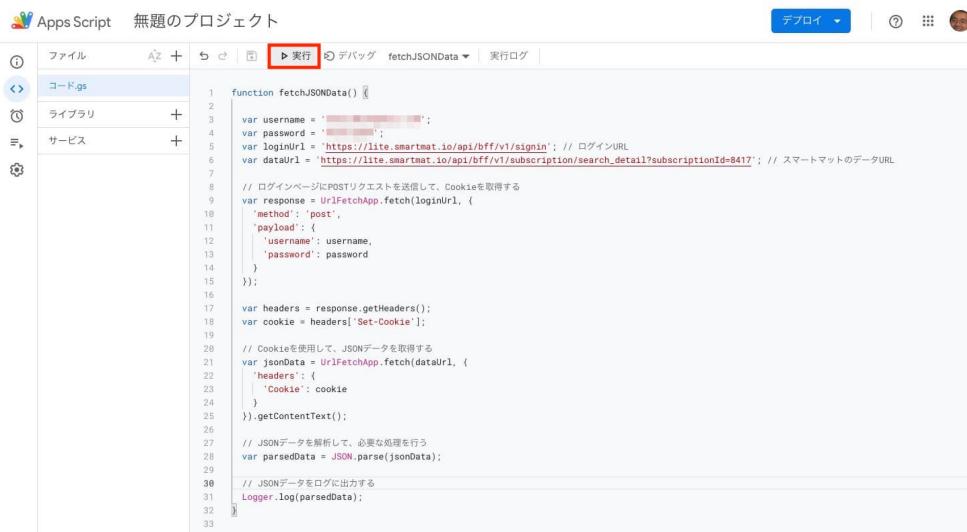


図 4.14: GAS の実行

初めて実行する場合、承認が必要ですので、「権限を確認」を押します。

第4章 スマートマットをハックしてさらに便利にする



図 4.15: 権限を確認

次に、アカウントを選択します。

4.8 GAS の実装

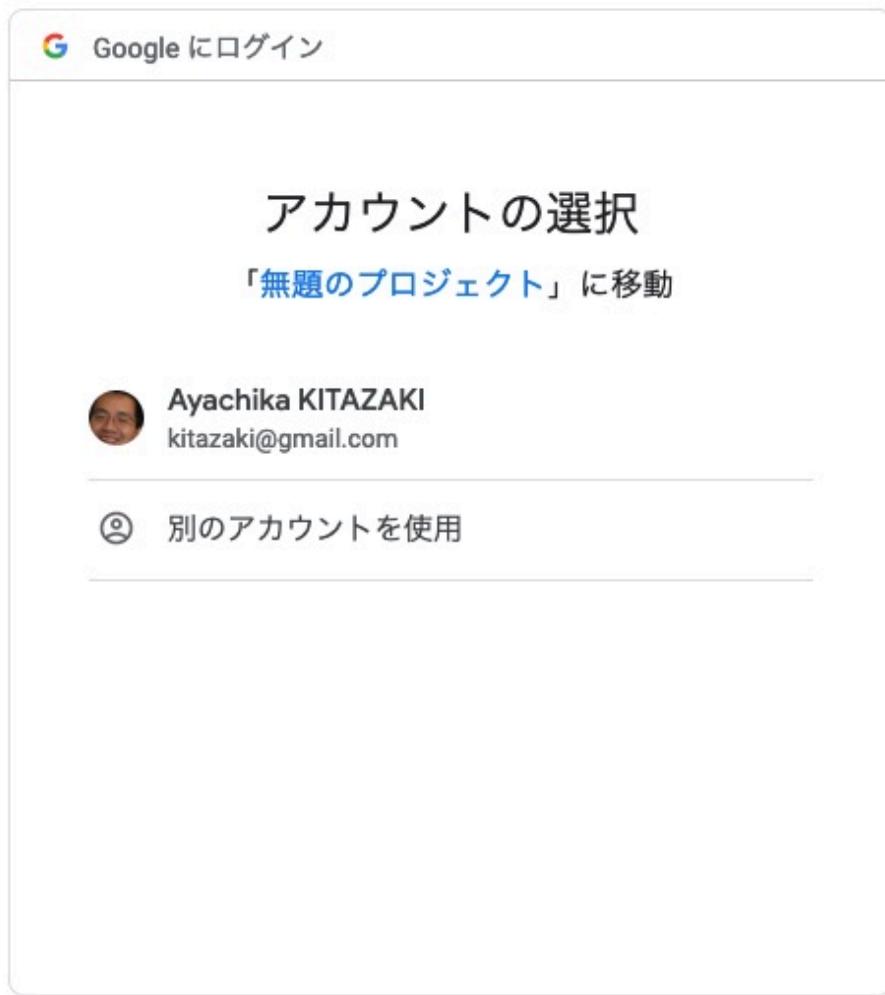


図 4.16: アカウントの選択

「詳細」を押したあと、「(安全ではないページ) に移動」を選択します。

第4章 スマートマットをハックしてさらに便利にする



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アクセスを求めてています。デベロッパー (kitazaki@gmail.com) と Google によって確認されるまで、このアプリを使用しないでください。

[詳細](#)

[安全なページに戻る](#)

図 4.17: 詳細の選択

リスクを理解し、デベロッパー (kitazaki@gmail.com) を信頼できる場合のみ、続行してください。

[無題のプロジェクト（安全ではないページ）に移動](#)

図 4.18: （安全ではないページ）に移動の選択

外部サービスへの接続で「許可」を押します。

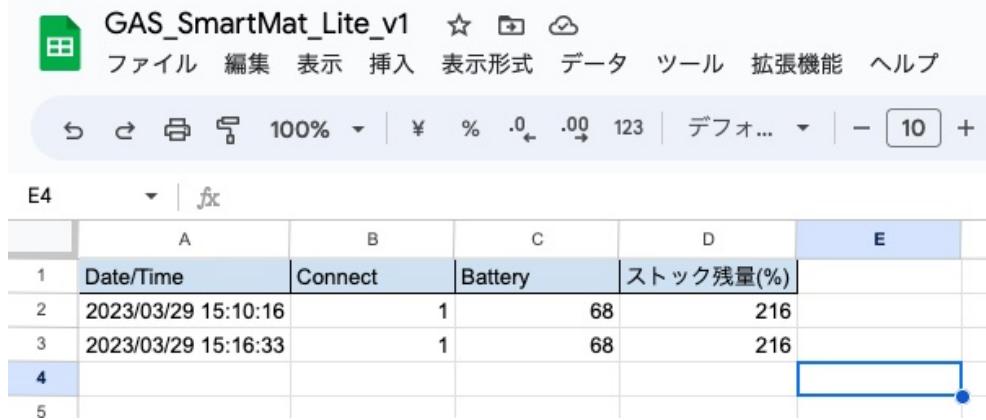
4.8 GAS の実装



図 4.19: 外部サービスへの接続

第4章 スマートマットをハックしてさらに便利にする

GAS の実行が正常に完了すると、スプレッドシートにスマートマットライトのデータが記録されるので、記録されたデータを用いてグラフを作成します。



1	Date/Time	Connect	Battery	ストック残量(%)
2	2023/03/29 15:10:16	1	68	216
3	2023/03/29 15:16:33	1	68	216
4				
5				

図 4.20: スプレッドシート

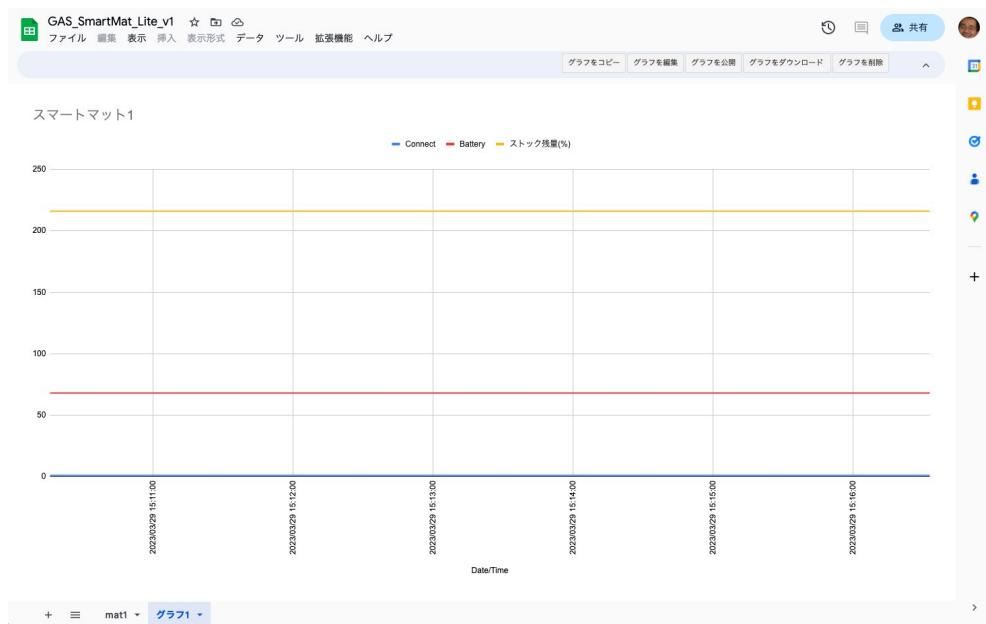


図 4.21: グラフ

GAS を定期実行する場合、トリガーを設定します。時計マークを押します。

4.8 GAS の実装



図 4.22: トリガーの設定

「トリガーを追加」を押します。

+ トリガーを追加

図 4.23: トリガーを追加

- ・「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」 → 時間ベースのタイマー
- ・「時間の間隔を選択 (時間)」 → 1 時間おき

を選択し、「保存」を押します。

第4章 スマートマットをハックしてさらに便利にする



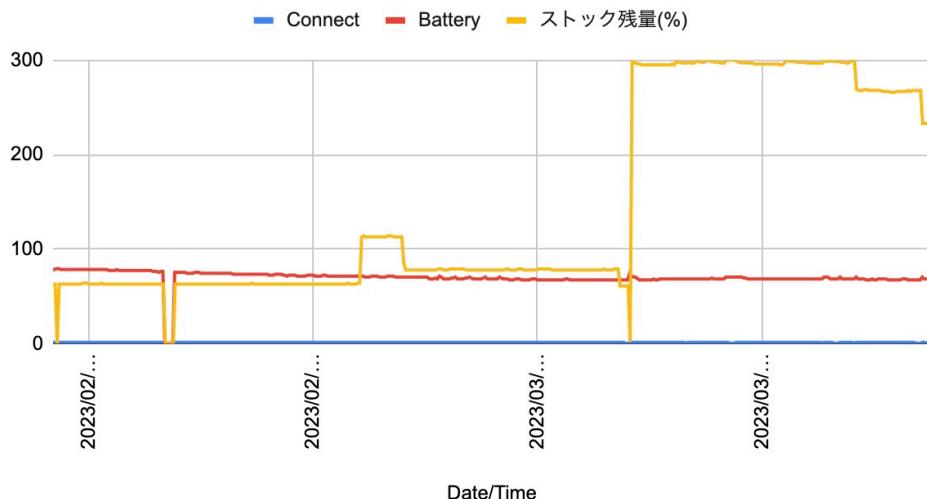
図 4.24: トリガーを保存

4.9 結果表示

在庫の消費傾向に加えて、Wi-Fi 接続状況と電池の消費傾向も把握できるようにしました。在庫が無くなる時期を予測したり、使い過ぎを警告することもできそうです。

4.9 結果表示

マット1



マット2

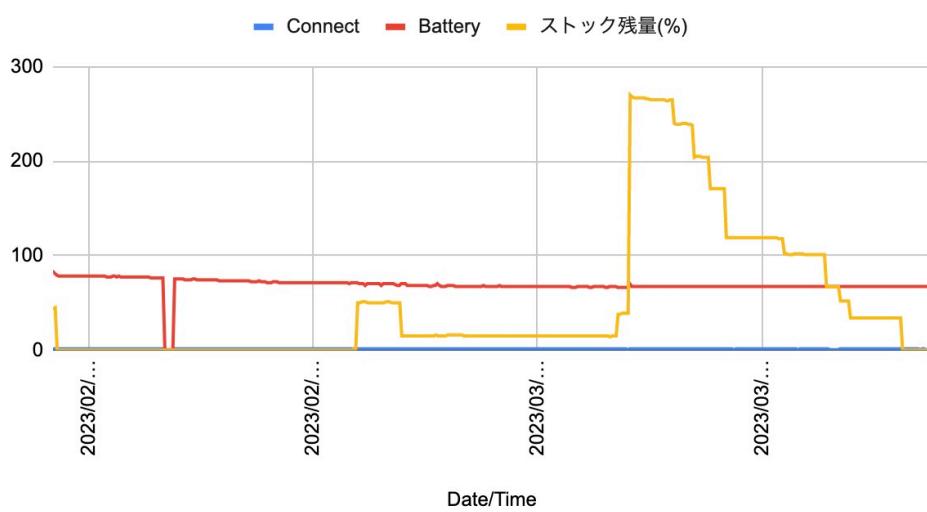


図 4.25: Wi-Fi 接続状況・バッテリー容量・ストック残量のグラフ

第4章 スマートマットをハックしてさらに便利にする

4.10 さいごに

スプレッドシートと GAS を使用して、スマートマットライトをハックしてみました。
みなさんもぜひ、いろんな IoT デバイスをハックして自動化を楽しんでみましょう！

第5章

RPAによるリアルタイム処理

鎌田 誠

5.1 RPA のメリット

RPA の導入目的として「生産性向上」と「品質向上」があります。24時間365日、文句も言わず動いてくれるRPAは、まさに生産性向上のために産まれてきた申し子のようなものですね。品質向上についても、ヒューマンエラーをしっかりと排除してくれて、常に一定の品質を確保できます。

一方、ここではあまり注視されていないRPAによる「リアルタイム処理」について語りたいと思います。

リアルタイム処理を実装することで、これまで実施できなかった業務ができるようになるなど、大きなメリットもありますので、是非最後までお読みいただければ嬉しく感じます。また本章ではPowerAutomate for desktopで作成したサンプルフローも掲載致します。その他のRPA製品でも同様のフローは作れますので、よかつたら参考にしてください。

5.2 RPA のリアルタイム処理とは？

5.2.1 リアルタイム処理を知る

製造業でのPRA導入事例として「作業日報の入力作業自動化」など、よくあがる事例です。

工場では、工程1→工程2→工程3→完成品のように、いくつかの工程を通して製品が生産されます。例えば工程2のYさんが機種X1を100台、機種X2を200台、機種X3を35台の作業をしたとしたら、その実績を作業日報に記入します。

第5章 RPAによるリアルタイム処理

作業日報						
部署名	工程2					
作業者名	Y					
日付	2023/4/18					
承認	審査	作成				
項目	加工指示番号	機種名	オーダー数	開始時間	終了時間	実数
1	WO-202304170010	X1	100	8:00	12:00	100
2	WO-202304170011	X2	200	13:00	16:00	200
3	WO-202304170012	X3	100	16:00	17:00	35 残数65は、明日対応
4						
5						
6						
7						
8						

図 5.1: 作業日報

各工程ごとにその作業日報を取りまとめて（例えば、工程2ではYさん含めて10名の方が作業をしていれば、10名分の作業日報）、基幹システムに投入します。作業日報ができるのは作業が終ってからですので、投入できるのは夕方か夜になりますよね。

そうすると、基幹システム上正しい在庫が見えるのは全ての投入が終ってからになります。つまり、翌朝にならないと基幹システム上では正しい在庫が見れないのです。

これは正しい姿でしょうか？

常に最新の「在庫状況」が基幹システム上で見えることで、最適な判断ができます。最新の在庫が基幹システム上で見えない以上、現場に赴き在庫状況を確認しなければならず、非常に無駄な作業となります。

しかし、現実はそこに人手を割いてでも最新にするのが難しく、結果、翌朝にならないと在庫が見えない状況になっている工場も多いです。

そこで、RPAによるリアルタイム処理が活きてきます。必要な時にシナリオを動かすのではなく、リアルタイム処理では常にシナリオが動き続け、必要な条件が揃った際に必要な処理をさせることができます。

先ほどの例では、Yさんが各機種の作業を完了した際にシナリオが自動的に動き始める仕組みです。これにより、常に最新の在庫状況が見れるようになります。

5.2 RPA のリアルタイム処理とは？

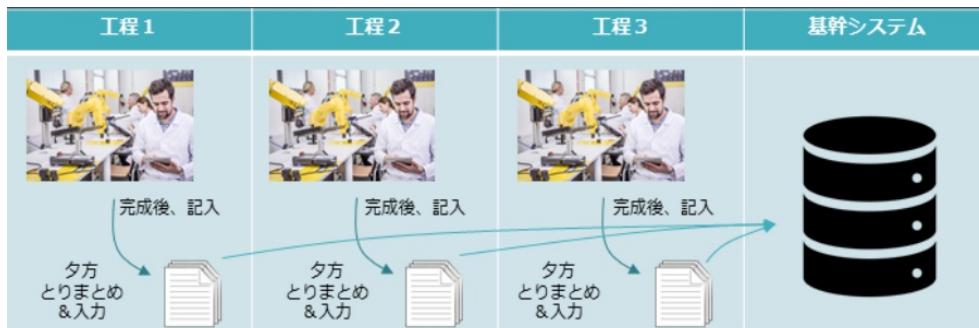


図 5.2: 工程移動フロー (改善前)

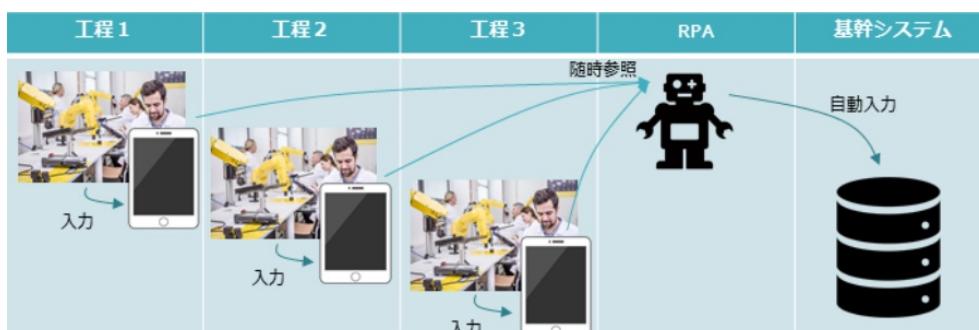


図 5.3: 工程移動フロー (改善後)

5.2.2 リアルタイム処理のメリット・デメリット

このように、本来実施したかったにも関わらず、人が実施するには現実的ではないような作業でも、RPA なら実現可能なのです。ただし、RPA でリアルタイム処理を実現させるためには、いくつかのメリット/デメリットがあります。

メリット

1. 常時動いているため、都度シナリオを実行する手間がない。
2. 必要なタイミングで自動で処理を実行してくれる
3. 処理 A, 処理 B 等複数の処理を埋め込むことで、1ライセンスでも複数処理が可能

デメリット

1. トリガーとなる電子的な仕組みが必要
2. シナリオが複雑化しやすく、エラー時の原因がわかりにくく

第5章 RPAによるリアルタイム処理

3. RPAでPCを占有してしまうため、専用のPCが必要となる

このようにみると、良いことばかりではありません。特にシナリオが複雑化しやすく、トラブル時の復旧時間が長くなりがちです。長くなる分、現場には負担を強いることになるので、可能な限りエラー原因が追跡できるような情報は残すようにしましょう。

5.3 リアルタイム処理の実装

リアルタイム処理を実装するには、処理が実行されるための条件が満たされるかどうかを常に監視する必要があります。そのため、無限ループを組んで、その中で発動条件を常に監視します。

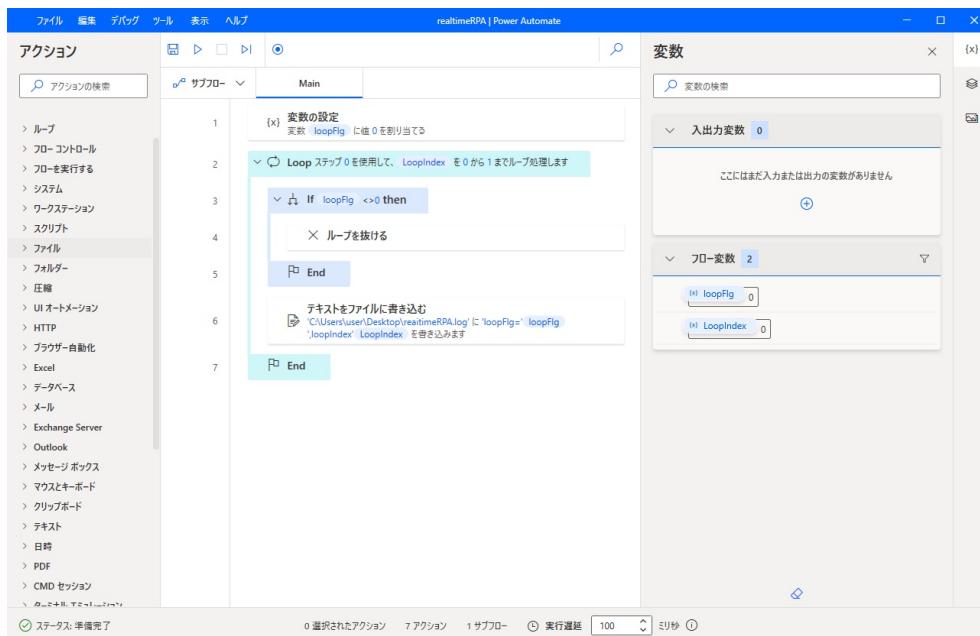


図 5.4: PowerAutomate for desktop による無限ループシナリオ

まず、loopFlg を初期化（0 を代入）します。次に、loopFlg が 0 以外の値になったらループを抜けるように Loop 処理を追加します。Loop の中では、loopFlg の値を確認し、0 以外の時にループを抜けるようにしています。また、Loop 内では、ログファイルに loopFlg の値と、loop 回数 (loopIndex) を出力しています。

この Loop 処理の中で、特定の条件が満たされた際に、所定の処理を実行するシナリオを作成すれば良いわけですが、特定の条件とは、どのようなものがあるでしょうか？ たとえば…

5.3 リアルタイム処理の実装

- 10時、12時、15時のような特定の時間を迎えた時
- 5分毎、10分間毎のような特定の周期を迎えた時
- あるファイルが生成された時

ぱっと思いつくのはこれぐらいでしょうか？

ここでは、「ファイルが生成された時」をトリガーとして処理するシナリオを考えていきましょう。

先ほどの作業日報の例で行くと、トリガーとしてはYさんの作業を完了させたタイミングになります。機種X1 100台分の作業が完了した際に、基幹システムにX1 100台と投入することで、在庫がリアルタイムで見えるようになります。そのためには、実績値を電子化する必要があります。自分は、データを入力するだけの簡単なフォームを作り、CSVファイルに出力するようにしました。今なら、M365のFormsを使ったり、チャットツールから入力し、PowerAutomateでCSVファイルを出力する、なんてやりかたもありそうですね。

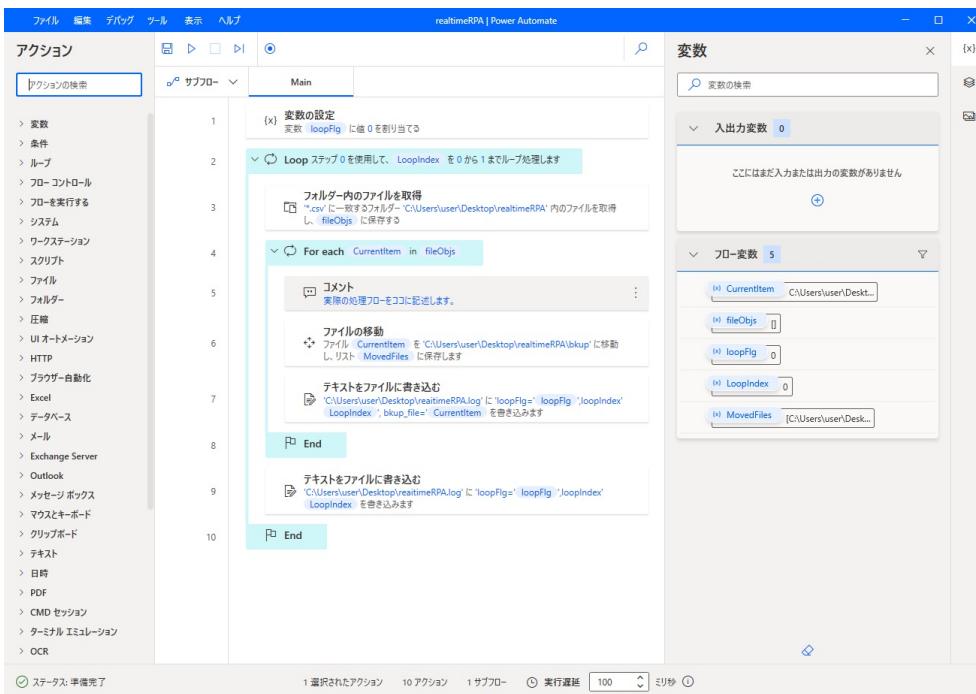


図 5.5: CSV ファイルが生成された際のロジック

ここでは、指定したフォルダ内に CSV ファイルがある場合に、コメント「実際の処理フローをココに記述します。」の箇所に処理フローを登録していきます。例えば、これま

第5章 RPAによるリアルタイム処理

での例のように作業日報の登録であれば、CSV ファイルを読み取り、基幹システムを操作して転記と登録を行います。併せて処理済みのファイルは bkup フォルダに移動させます。これは同じファイルで二重登録を避ける目的と、実際に登録したデータのエビデンスを残す意味があります。またログファイルも出力しております。ログファイルについては後ほど説明致します。

このようなフローを組むことで、基幹システムへの登録作業を RPA が人知れずリアルタイムに行ってくれることで、これまでどうしても実現できなかったリアルタイムでの正しい在庫管理ができるようになりました。

5.4 リアルタイム処理の適用事例

RPA におけるリアルタイム処理の事例を見てみましょう。

5.4.1 製造業における工程間在庫移動

これまでの事例としてあげてきたものが、製造業における在庫移動の事例になります。実際に私が情シス時代に構築したシナリオで、数値もかなりリアルな値になっています。では改めて処理フローを見てみましょう。

5.4 リアルタイム処理の適用事例

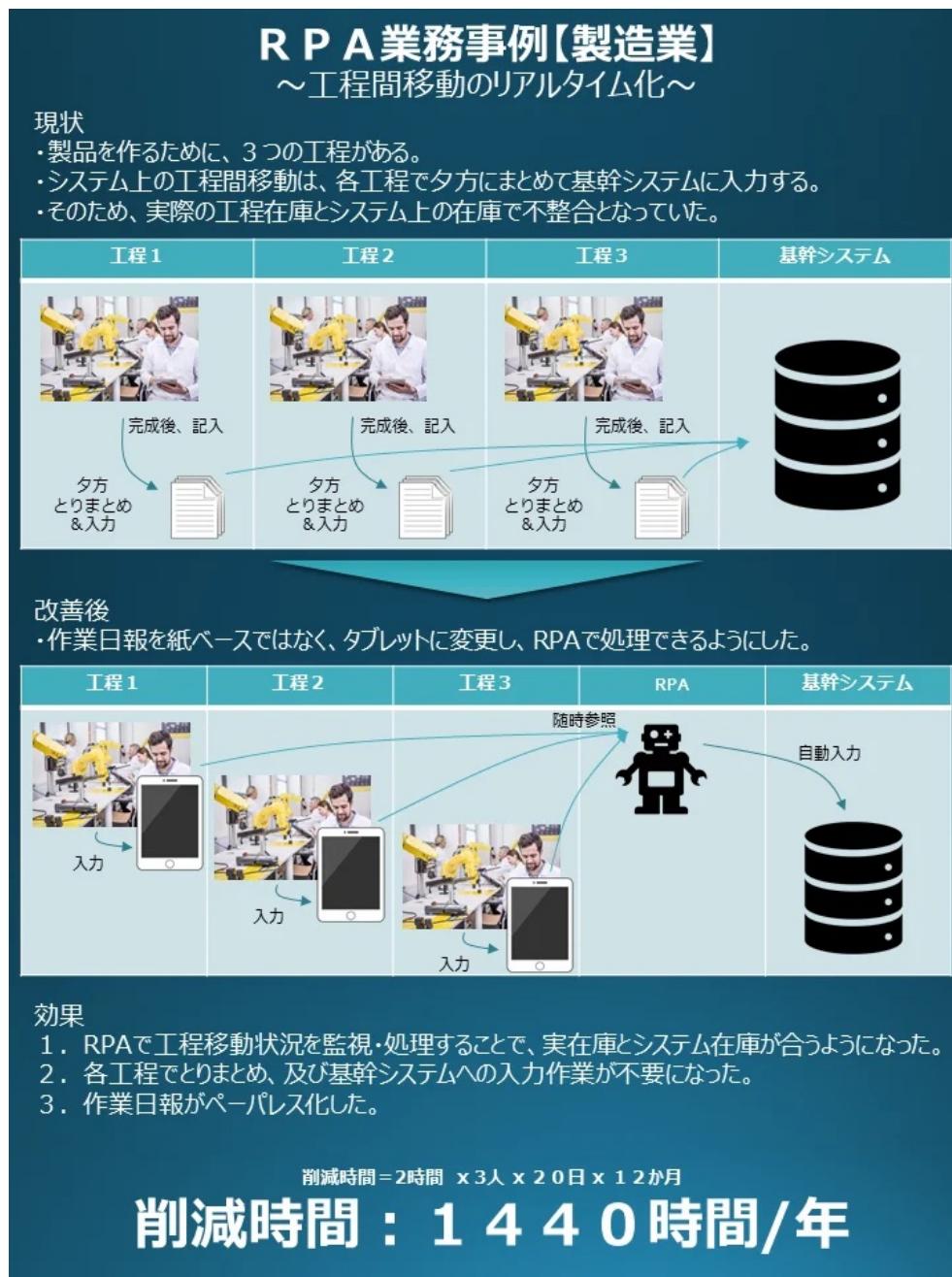


図 5.6: 工程間移動の実例

5.4.2 製造業における完成品入力

前項は在庫間移動でしたが、ここでは完成品入力になります。製造業においては、1つの機種を生産するのに、加工指示書と呼ばれる書類が発行されます。各現場は、この加工指示書に基づいて実際の生産を実施します。こちらも、私が情シス時代に実際に構築したシナリオです。完成品入力は、現場応援のため総務部門で入力することになっており、加工指示書と作業日報の物理的な移動（現場→総務）も発生していました。

5.4 リアルタイム処理の適用事例

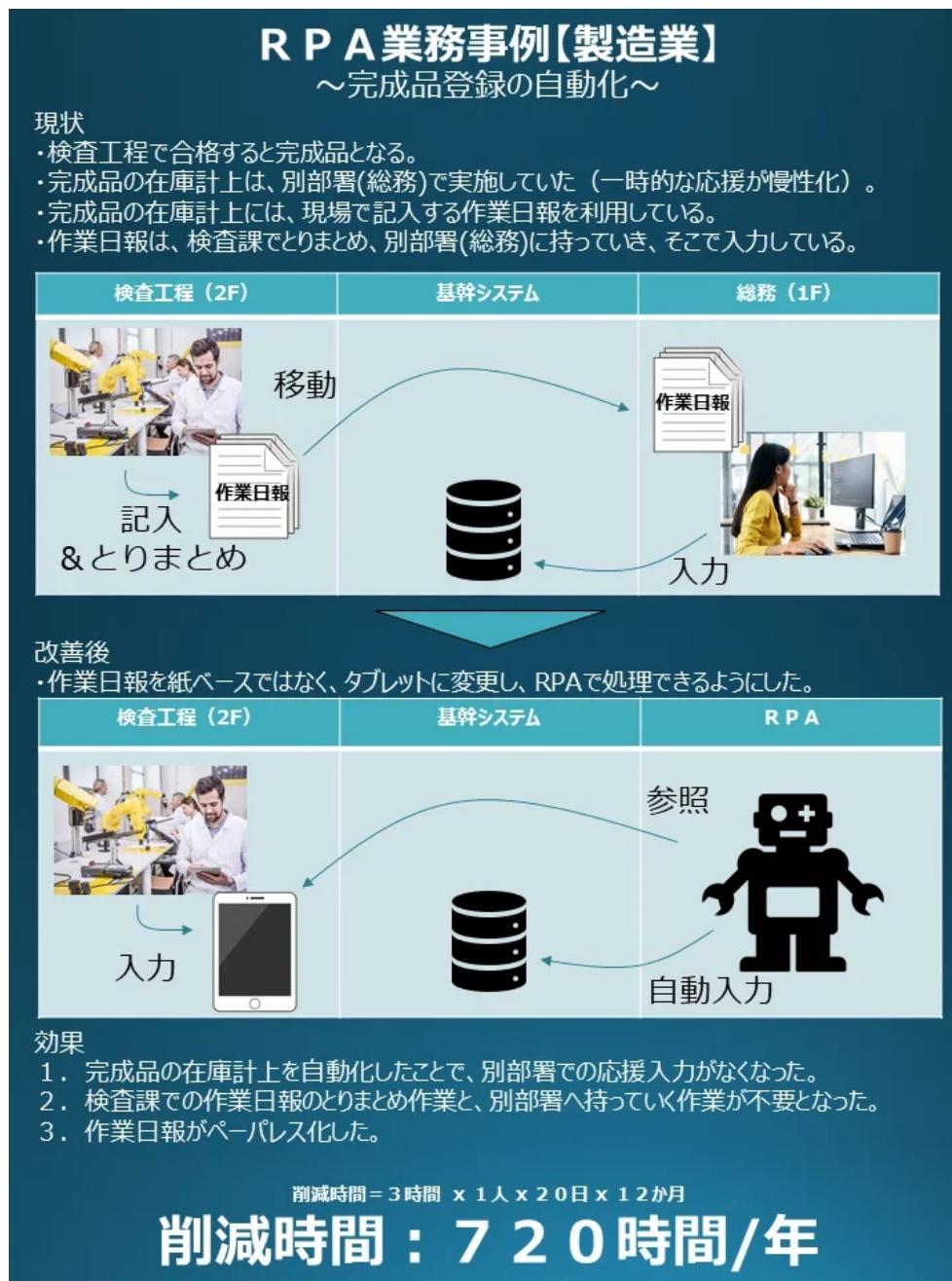


図 5.7: 完成品登録の事例

5.5 リアルタイム処理の課題と解決策

リアルタイム処理は、パソコンを占有してしまうため、常に人が見続ける（エラーなく処理が終わるか監視する）ことができません。そのため、エラーが発生した際に如何に早く気づくことができるか、またなぜエラーが発生したのかを知ることが大切です。

5.5.1 エラーを素早く知るには

リアルタイム処理で動くシナリオは、通常、人が監視をしていないところで動きます。よって、異常終了等でシナリオが止まっていても知ることができず、現場の方からの指摘で知ることになってしまいます。そうならないためには、エラー時はしっかりとエラーをトラップし、管理者へ通知することが大切です。また、通知も一人だけではなく、関係者複数人に送るようにしましょう。

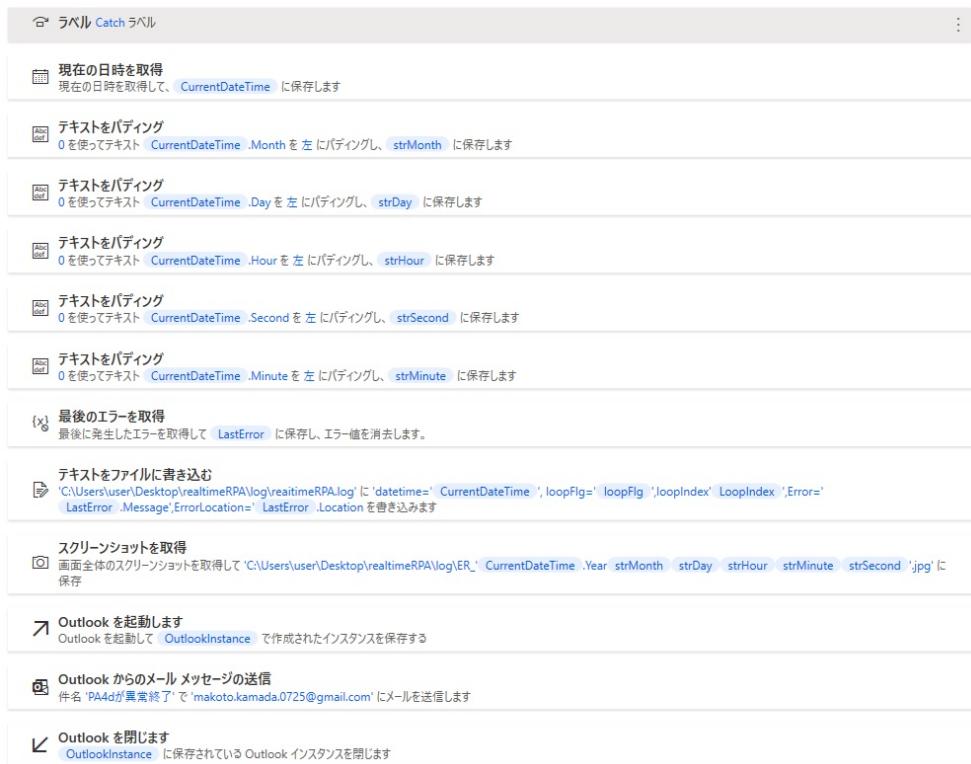


図 5.8: エラー時の処理例

5.5 リアルタイム処理の課題と解決策

こちらのフローは、エラー発生時に処理されるフローになります。ログファイルへの書き込みと、画面キャプチャを取得し、Outlook を使ってメール送信をおこなっています。今回はメールとしましたが、Teams 等のチャットツールに送る方が、より早く周知できるメリットがあります。

5.5.2 エラーの原因を知るには

エラーでシナリオが止まってしまった場合を想定して、シナリオのどこまで進んだのか、その際の変数の状況はどうだったのかをファイルに保存しておくようにしましょう。また可能であれば、エラー時の画面キャプチャも保存しておくと、ログだけでは分からぬ状況が見え、解決のためのヒントとなる場合も多いのでオススメです。

実際にエラーで止まってしまった場合は、取得したログやキャプチャ画面等をヒントに原因を特定し対処していくことになります。そのために、ログファイルに記録すべき情報を検討します。一般的には、下記のような項目になります。

ログの種類

- シナリオ名
- 発生日時
- 直前の処理名（行番号など）
- エラーメッセージ
- 変数の値

などなど

図 5.9 の左側は、図 5.8 で示したフローにあるログ情報をログファイルに保存するノードの画面です。一方、右側はエラー発生時のスクリーンショットの記録です。

第5章 RPAによるリアルタイム処理

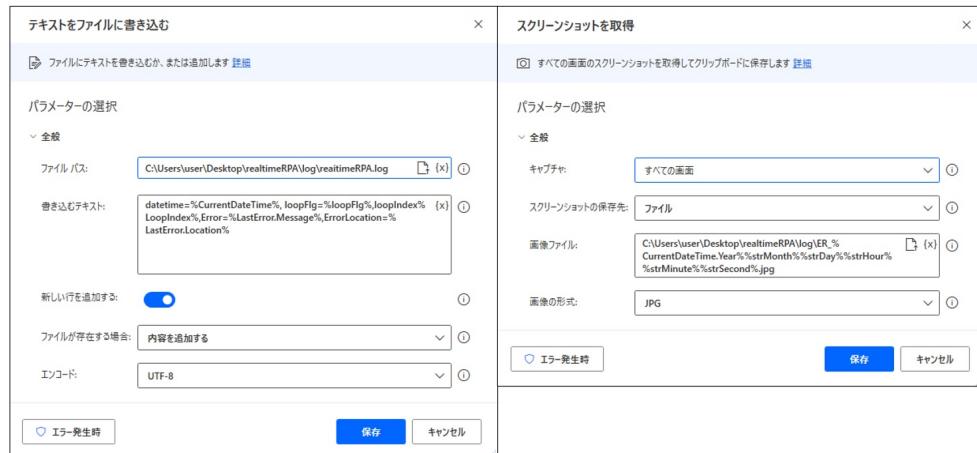


図 5.9: エラー時のログ取得

シナリオをリリースする前に、トラブル対応に必要な情報（ログ）が十分に出力されているか確認しておきましょう。

また、登録が目的のフローであれば、登録がどこまで済んでいたのかの確認も必要となります。実際の登録状況と、監視フォルダや bkp フォルダ、またログファイルの内容を見比べ、どこから処理を再開させれば良いのかを判断する必要があります。

5.6 リアルタイム処理の今後の展望

RPAにおいては、生産性向上、品質向上の2点に注目が集められていましたが、このようにRPAを活用することで、人手ではできなかったあるべき姿に近づくことができます。全てのシステムがクラウド化されれば、クラウドフローで事足りることかもしれません、まだまだオンプレの仕組みも多いですし、クラウドからオンプレ回帰の流れもできます。また、AIによる自動化等もRPAには追い風だと思っています。

5.7 最後に

今回記載した内容は、一部 note記事にしてあったり、過去のRPACommunityのLT会でも発表した内容になりますので、併せてそちらも参照頂けると幸いです。

note

RPA事例集について

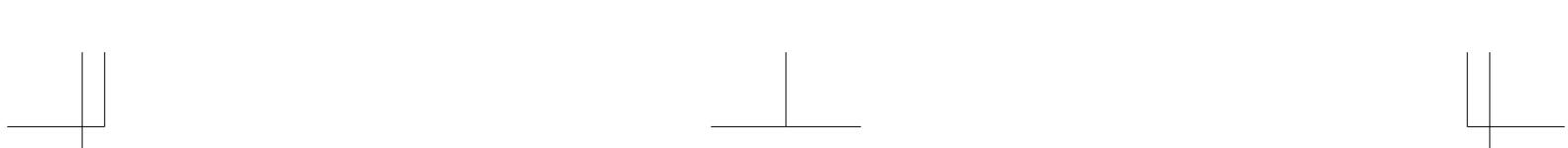
<https://note.com/kerdy/n/nd2f20dad64f2>

Youtube

5.7 最後に

RPA でリアルタイム処理

<https://youtu.be/MMfycoq9tsc?t=789>



第 6 章

Raspberry Pi と GCP を使って、 SMS の E2E 監視を実装してみた！

瀧川 大樹

6.1 はじめに

こんにちは！ たつきちです。私は普段、某通信事業者のショートメッセージングサービス（以下 SMS と記載）を提供するシステムの保守業務をしています。本章では Raspberry Pi を使って SMS の送受信テストを自動化し、GCP の GAE と Cloud SQL で WebAPI を構築して、社内の監視システムに連携させた事例について紹介します。

6.2 E2E 監視の重要性と構築した経緯

なじみがない方のために、SMS について少し紹介させて頂きます。SMS は電話番号を送信先情報に指定し、テキストメッセージをやり取りするメッセージングサービスです。ユーザコミュニケーションの用途としての利用は減っていますが、Web サービスの 2 段階認証の際のワンタイムパスワードをユーザへ送付するような、システムからユーザに送られるケースについての利用は増えています。社内では SMS 関連のシステムは重要なインフラとして定義され、障害が発生した場合については、迅速な状況確認と復旧が要求されます。私たち運用担当は障害対応の際、まず、お客様が実際にサービスを使っているかどうかを最初に確認します。ここで、お客様が実際にサービスを使っているのか確認するために、E2E 監視は非常に重要です。

E2E 監視とは、システム外部からユーザと同様の方法でアクセスして、そのシステムが正常に稼働しているか確認するための監視です。障害が発生した際に手作業で端末を使って試験をしていると障害検知・復旧が遅くなり、お客様と約束している品質を守る事が難

第6章 Raspberry Pi と GCP を使って、SMS の E2E 監視を実装してみた!

しくなります。そのため、私たちの部署では、システムの導入の際には、E2E 監視の実装をする事を必須の条件としています。E2E 監視の実装の内容としては、サンプルとしてお客様のリクエストをエンドポイントに実行し、その応答結果や応答時間を監視しています。

以前まで、SMS の E2E 監視は、SMS の E2E 監視専用のサーバを構築して実現していました。SMS が利用するプロトコルは非常にニッチな事もあり、SMS の E2E 監視専用のサーバはベンダーが構築したものを利用していました。最近、この SMS の E2E 監視サーバのサポート期間が終了することになり、構築をしたベンダーからリプレイスの提案を頂きました。その提案頂いた費用が非常に高額であったため、自社内製で SMS の E2E 監視を実装する検討を開始しました。

6.3 監視の構成

E2E 監視システムは、SMS の送受信を行う SMS 送受信シミュレータ、監視結果を表示する監視ダッシュボード、シミュレータと監視ダッシュボードを連携させる Web API の 3 つで構成されています。以下、それぞれについて少し詳しく説明します。

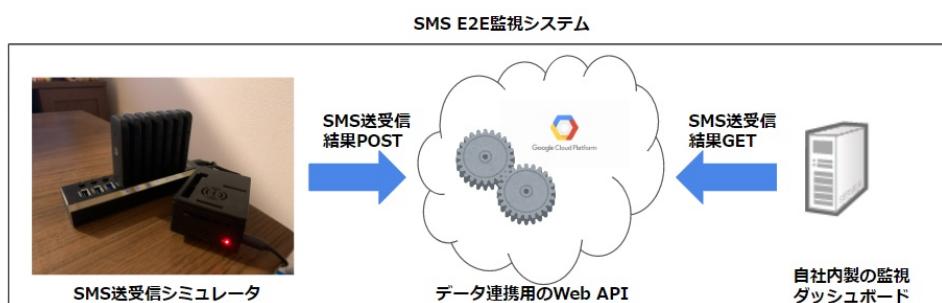


図 6.1: E2E 監視システムの構成

6.3.1 SMS 送受信シミュレータ

SMS の E2E 監視を実現するために、SMS の送信・受信をシミュレートする方法を検討しネットで調べたところ、Raspberry Pi^{*1}と USB 接続の LTE ドングルを使って SMS を送受信を試しているネット記事を見つけました。Raspberry Pi とは、教育目的に製造された小型コンピュータで、現在普及が進んでおり教育分野以外も様々な分野で利用されて

*1 https://ja.wikipedia.org/wiki/Raspberry_Pi

6.3 監視の構成

います。

また、USB 接続の LTE ドングルは、外出した際に Wifi 環境がない場合に利用する USB 型のモデムの事を指します。Raspberry Pi から AT コマンドと呼ばれるモデムを操作するコマンドをシリアル接続でドングルに流し込む事により、通話やインターネット接続や SMS の送受信などの携帯端末の操作を実行する事ができます。今回は、AT コマンドを使った SMS の送受信の処理をスクリプトとして実装する事によって、SMS 送受信シミュレータを実現しました。詳しくは実装パートで説明します。



図 6.2: SMS 送受信シミュレータ

6.3.2 監視ダッシュボード

私の属している部署で運用・保守しているサービスは、自社内製の監視ダッシュボードで E2E の監視の状態を確認できるようになっています。監視の方式は統一されている事が望ましいと考えたため、SMS 送受信シミュレータの送受信結果は監視ダッシュボード上に表示させる事にしました。ダッシュボードは社内の商用ネットワークに属しており、SMS 送受信シミュレータとダッシュボードを直接連携させるのは、セキュリティ上避

第6章 Raspberry Pi と GCP を使って、SMS の E2E 監視を実装してみた!

けたいと考えました。連携については、後述するデータ連携用の Web API を利用して連携するような方式としました。

6.3.3 データ連携用の Web API

シミュレータでの SMS の送受信結果をダッシュボードに表示させるために、部署で契約している Google Cloud Platform(GCP) 上に Web API を構築しました。社内でのクラウド利用については、2つの要件を満たせば利用が可能となります。Cloud 環境を社内で使ってみたいと考えている方は、実装の前に、会社の利用ルールや部門の予算の状況を確認する事をおすすめします。

6.3.4 予算の確保

クラウド利用に当たって、部門でクラウド利用のための予算を取っているため、利用に当たってどのくらいお金が当たるかを部門内で説明が必要でした。

6.3.5 社内のセキュリティポリシーを満たす事

実装に当たっては社内のセキュリティ部門が提供するセキュリティガイドラインを満たすように実装する必要があります。また、実装後に脆弱性診断を受け、診断結果が問題なければ実装したものを使って運用開始することができます。また、運用を開始してからも年次で脆弱性診断を実施する必要と月次でセキュリティパッチの適用調査をする必要があります。

6.4 SMS 送受信シミュレータの実装

SMS 送受信シミュレータの要件は大きく 2 つあります。1 つ目は、SMS の送受信を実行できる事、2 つ目はインターネットに接続し、SMS の送受信結果を Web API を使って登録できることです。

6.5 SMS 送受信スクリプトの処理概要

1 つ目の要件を満たすために、SMS 送受信スクリプトを実装しました。SMS 送受信スクリプトの処理の概要について説明します。

6.5 SMS 送受信スクリプトの処理概要

6.5.1 AT コマンド投入用の USB デバイスファイルの認識

ドングルを Raspberry Pi に接続すると、デバイスファイルが複数作成されます。スクリプトからシリアル接続する際にライブラリでエラーが出力されるため、wvdial^{*2}用と AT コマンド登録用で USB デバイスファイルを分ける必要があります。そのためスクリプト上では、AT コマンド投入用のデバイスファイルのパスを定義しています。複数のドングルを接続すると、デバイスファイルとドングルの紐づけができなくなるため、デバイスファイルを固定化する対応が必要となります。具体的には、udev ルールに定義する事によってデバイスファイルを任意の名前に固定化することができます。USB の接続ポートやドングルのシリアルの情報で固定化できるのですが、利用しているドングルはシリアルの情報がなかったため、USB 接続ポートで固定化を実施しています。

6.5.2 SMS の送受信

SMS を送受信する前に、SMS の送受信できる状態であるかどうか、ネットワーク状態(以下 NW 状態と記載)の判定を実施しています。NW 状態の判定には、AT コマンドで電波強度確認、在圏状態の確認を実施しています。また、SMS の送受信を可能にするために、SMS モード設定コマンド等を実施しています。ドングルで SMS を送受信する際に注意が必要な点は 2 つあり、1 つ目は SMS の文字コードで、正しく設定しないと文字化けをします。実装したシミュレータでは GSM に設定しています。2 つ目は SMS の読み込み先、書き込み先の設定です。読み書きでそれぞれ端末か SIM カードで保存するかを設定する事ができますが、書き込み先と読み込み先を一致させる必要があります。今回の場合ドングル上か SIM カード上になりますが、受信したら都度削除するシナリオになっているので、SIM 上を保存先として指定しています。

6.5.3 Web API で送受信結果を Post

SMS の送受信の試行を実施後、Web API のユーザとパスワードで認証トークンを取得し、認証トークンで SMS の送受信結果を POST します。工夫したところは WebAPI のユーザとパスワードの秘匿性を上げるために、パスワード管理モジュールを使って、スクリプト内に認証情報を直接埋め込まないようにしています。スクリプト内にパスワード等の情報を埋め込んでしまうとスクリプトが誤って外部に公開された場合、認証が危険化するリスクがあると考えたため、実装はやや煩雑になりましたが対策を実施しました。

^{*2} <https://wiki.archlinux.jp/index.php/Wvdial>

6.6 インターネット接続処理の概要

2つめの要件を満たすために wvdial と呼ばれる Point-to-Point Protocol ダイアラをインストールしてインターネットへの接続を可能としています。インターネットへの接続には、wvdial の設定ファイルの編集と AT コマンドを使ってドングルへの設定投入が必要でした。設定ファイルには、ドングルのデバイスファイルのパス、ドングルに搭載した SIM の APN の情報を設定します。次に、ドングルへの設定についてですが、ドングルに AT コマンドを投入して通信事業者のネットワークに接続します。設定に必要な AT コマンドを「6.7.1 APN の設定」に記載しています。シミュレータは Quectel 製の LTE チップが実装された、SORACOM Onyx LTE USB ドングル^{*3}を利用しておらず、「6.7.2 APN のユーザとパスワードの設定」と「6.7.3 Packet Data Protocol(PDP) 有効化」については、Quectel 製のオリジナルのコマンドとなります。

6.7 ドングルに設定必要な AT コマンド

6.7.1 APN の設定

AT+CGDCOUNT=1,"IP", "利用する APN 名"

6.7.2 APN のユーザとパスワードの設定

AT+QICSGP=1,1,"APN 名", "ユーザ", "パスワード", 0

6.7.3 Packet Data Protocol(PDP) 有効化

AT+QIACT=1

6.8 WebAPI の実装

Web API は GCP 上の Google App Engine(GAE)^{*4}と Cloud SQL^{*5}を使って実装しました。運用の負荷をかけずに、実装は柔軟にしたいという考え方で PaaS を利用して実装しました。GAE は認証 API と登録 API と結果取得 API を提供しています。また、Cloud SQL にはシミュレータで実行した SMS の配信結果情報が登録 API で保存されています。インターネット上に公開しているため、以下のようなセキュリティの対策を行っ

^{*3} <https://soracom.jp/store/7326/>

^{*4} <https://cloud.google.com/appengine?hl=ja>

^{*5} <https://cloud.google.com/sql?hl=ja>

6.8 WebAPI の実装

ています。

6.8.1 ファイアウォールで接続 IP を制限

GAE はファイアウォールの機能を提供しており、GAE 上にアクセスする IP を制限することができます。今回接続するシミュレータの IP とダッシュボードの IP に接続を制限しています。接続するシミュレータの IP は事業者毎によって振られるものとなるため、広いレンジで制限をかけています。

The screenshot shows the 'Firewall Rules' section of the App Engine dashboard. On the left sidebar, 'Firewall Rules' is selected under the 'App Engine' heading. The main area displays a table of firewall rules:

優先度 ↑	アクション	IP範囲	説明
500	許可		⋮
800	許可		⋮
900	許可		⋮
1000	許可		⋮
1200	許可		⋮
デフォルト	拒否		⋮

図 6.3: ファイアウォールルール画面

6.8.2 認証キーの提供と有効期限の短時間化

結果取得 API、認証 API を実行するためには、認証 API で取得した認証キーを必要とするように実装しています。また、そのキーの有効期限については非常に短い期間に設定しています。

6.8.3 入力値のバリデーション機能とプレースホルダの利用

SQL インジェクションの対策として、スクリプトではバリデーション機能とプレースホルダを実装しています。バリデーションは公開されているオープンソースのライブラリを利用して実装しています。API の入力値についてルールを設定し、ルールに違反する入力値に関しては、エラー応答を返すようにしています。プレースホルダについても公開されているオープンソースのライブラリを利用して実装しています。入力値内に特殊文字が

第6章 Raspberry Pi と GCP を使って、SMS の E2E 監視を実装してみた!

あればエスケープを実施し、想定外の SQL が実行される事を防ぎます。

6.8.4 リクエスト数とリクエストエラーの監視

GCP が提供する Cloud Monitoring^{*6}でリクエストエラーとリクエスト数を監視しています。Cloud Monitoring では API の実行ログやメトリクスを確認できます。また、Cloud Monitoring が提供するアラート機能によって、メトリクス等に閾値を設定し、リクエストのエラーやリクエスト数が設定した閾値を超過した場合は、メール等で異常を知る事ができます。

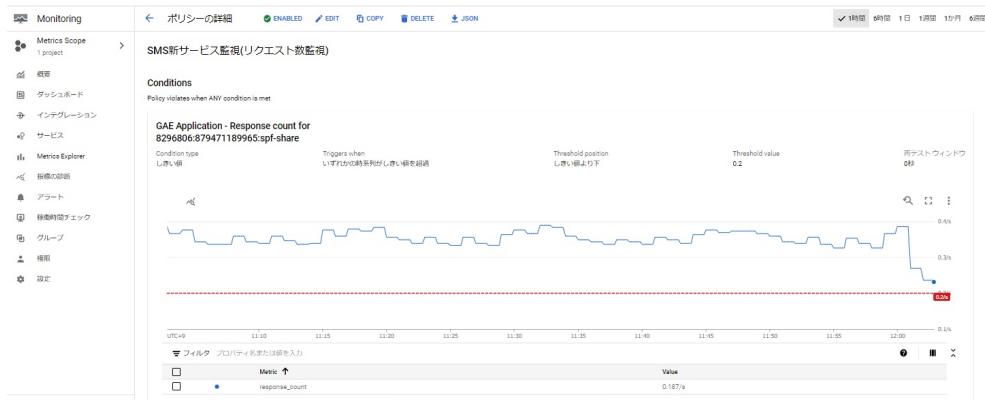


図 6.4: アラート機能画面

6.9 自社内製の監視ダッシュボード側の実装

シミュレータと WebAPI については自身で実装をしましたが、自社内製の監視ダッシュボードの実装については、実装をする専門の部隊がいたため、実装を依頼しました。監視ダッシュボードに実装が必要な機能は、定期的に結果取得 API で SMS の配信結果を取得し、ダッシュボード上に OK or NG を表示させるというものです。実装を専門部隊にお願いするにあたって、E2E 監視の構成や API の仕様書を起こして、要件を実装部隊に伝える必要がありました。E2E 監視の構成や API の入出力の情報や期待する処理フロー等をパワポで作成して要件を伝えて、無事に期限内に実装がされました。今回、実装を依頼する際に API の仕様をパワポにわざわざ起こして実装を依頼したのですが、Google 先生に聞いてみると Swagger^{*7}と呼ばれる、Web API 開発環境がある事を知り

*6 <https://cloud.google.com/monitoring?hl=ja>

*7 <https://swagger.io/>

6.10 初期費用とランニング費用

ました。Swagger を使えば Web API の設計、ドキュメントの自動生成、テスト等が効率良くできそうなので、今後 Web API を構築する機会があれば使ってみたいと思います。

6.10 初期費用とランニング費用

構築と運用にかかる費用としては、以下の初期費用とランニングコストが発生します。基本料と SMS 送信料金のランニングコストについては月 6 万円程かかってます。現状、ランニングコストに月 10 万円程かかっていますが、ランニングコストの累計額が、ベンダーから提示されたリプレイスの見積もりに達するのは、何百年先になるので費用効果については十分あったと考えています。

初期費用

- シミュレータの実装費用
- SIM 初期契約料

ランニング費用

- GAE と Cloud SQL の費用
- 通信事業者の基本料 + データ使用料
- SMS の送信費用

以下、それぞれの詳細について紹介いたします。

6.11 初期費用について

シミュレータは Raspberry Pi とドングルの費用で 1 台大体 2 万円ぐらいかかります。半導体不足の影響で Raspberry Pi の値段が上がっているようなので、今はもう少しかかるかもしれません。SIM の初期契約料については、MVNO のソラコムさんが提供している特定地域向け IoT SIM を契約していて、初期費用は DCM の SIM で 3300 円、AU の SIM で 1650 円でした。

6.12 ランニング費用について

6.12.1 GAE と Cloud SQL の費用

GAE については、安定性と不要に料金追加を防ぐためにインスタンスについてはオーバースケールではなく、固定化していて、念のためインスタンス数は 2 つにして冗長化しています。

Cloud SQL に関しては処理も多くなく、ストレージの容量も必要ないので一番低いス

第6章 Raspberry Pi と GCP を使って、SMS の E2E 監視を実装してみた!

ペックとしました。クラウド側のランニングコストに関しては、1ヵ月に GAE が 2 万円、Cloud SQL が 8 千円程となっています。GAE と Cloud SQL のスペックについては以下の通りです。約 0.5rec/s を処理しています。

GAE

- 環境:Standard
- リージョン:asia-northeast1
- インスタンス数:2
- インスタンスクラス:B1

Cloud SQL

- リージョン : asia-northeast1
- DB : MySQL
- マシンタイプ : 標準
- vCPU : 1
- メモリ : 3.75
- ストレージ : 10G

6.12.2 通信事業者の基本料 + データ使用料

通信事業者の基本料とデータ使用料については、月 300MB 分のデータ使用量が基本料に含まれ、超過した際は追加でデータ料を支払う料金体系となっていて、契約した SIM につき数百円/月の料金が発生します。

6.12.3 SMS の送信費用

SMS の送信費用については、自事業者同士の SMS 送信に関しては無料ですが、異なる事業者同士の SMS 送信については、費用が発生してしまいます。SMS の送信については 1 通 3 円かかり、サービス仕様上の 1 日に送信できる通数の制限が 200 通のため、上限にからないように送信をしています。

6.13 運用について

SMS E2E 監視の運用について説明します。監視システムに関わるコンポーネントがシミュレータ、GCP、監視ダッシュボードが多い事から誤検知が多くなり、従来より監視品質が低下する事が想定されました。そのため、以下の 3 つの対処を行う事によって、誤検知か障害かを切り分けています。

6.14 まとめ

6.13.1 対処 1: 複数の監視シナリオで監視する。

シミュレータに異常が発生するケースに対する対処です。2台のシミュレータで監視シナリオを2つ用意しておき、2つの監視シナリオがNGとなった際は障害と扱うように運用フローとして立てつけました。この対処により、シミュレータ故障による障害の切り分けが可能となります。

6.13.2 対処 2: 複数の監視手段を用意する

Web API側に異常が発生するケースに対する対処です。シミュレータ⇒Web API⇒監視ダッシュボードとは別の監視手段として、シミュレータからのSMSの送信・受信のログを監視しシステムのアラームとして発報させるようにしています。監視ダッシュボード上NGかつ上記のログ監視のアラームが発報した場合は障害と扱うような運用フローを立てています。

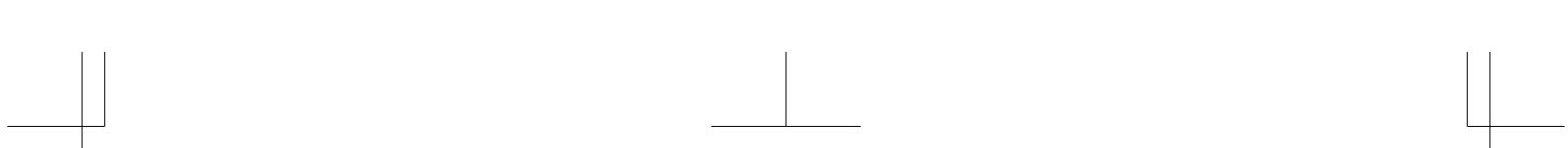
6.13.3 対処 3: 複数のネットワークからAPIを叩く

監視ダッシュボードに異常が発生するケースに対する対処です。監視ダッシュボードとは別に誤検知確認ツールをGoogle Apps Scriptで実装しています。誤検知検知ツールは監視ダッシュボードと仕様は同じでSMSの配信結果取得APIを定期的に叩いて、その結果をスプレッドシート上に表示させるようにしています。監視ダッシュボードと誤検知確認ツール双方NGの場合障害と扱うようにしています。

運用は少し煩雑となりますが、複数のパターンを用意しておき、組み合わせによって監視の確度を上げる対策をしています。E2E監視を入れていない場合は、システム監視担当が実端末を使って手動でSMS送受信のテストを実施していましたが、E2E監視を導入する事によって、手動でのSMS送受信テストを実施せずに良くなりましたが、現場の工数削減にも寄与できたと考えています。

6.14 まとめ

今回、Raspberry Piを利用したSMSサービスのE2E監視を実装した経緯、システム構成、システムの実装、運用についてご紹介させて頂きました。シミュレータの実装、GCPの機能、構築の費用、監視・運用等の雑多な内容になってしましましたが、何か一つでも困りごとを解決する糸口になれば幸いです。



第7章

現場で使える Python 自動化入門

青木 敬樹

7.1 はじめに

これを手に取られた方は実務で使用する自動化に興味がある方でしょうか？この章は Python 自動化入門としています。しかし、一般書でも様々な自動化の手法やライブラリの紹介が豊富にされています。そのため本章では、Python を実務で使用した際の私の経験に基づく Tips を詰め込みたいと思います。Python 限定の話もありますが、言語に特定されず別のもので読み変えていただく事もできると思います。本章は

1. 頭をすっきりテスト駆動開発
2. みんなで読もう Sphinx
3. 最後の救い log 取得

の 3 つで構成されています。

7.2 頭をすっきりテスト駆動開発

7.2.1 はじめに

皆さんはテスト駆動開発という言葉をご存知でしょうか？アジャイル開発で有名なこちらのメソッド、提唱したのは Kent Beck です。

まず初めにテスト駆動開発を知る前のこのメソッドに対するイメージは「面倒くさそう」というものでした。これは個人開発程度であればテストという行為そのものが必要ではなかった為であり、また成果物の品質を担保するためのテストとテスト駆動開発を混同していた為です。そのため開発後の工程を取り入れる必要性が判りませんでした。しか

第7章 現場で使える Python 自動化入門

し、今ではその恩恵にあづかっておりそのイメージは「多少手間はかかるが便利で有効」と変わっています。この説ではテスト駆動のメリットとその手法の紹介を行ないたいと思います。

7.2.2 テスト駆動とは？

まずテスト駆動開発自体の紹介を行なう前に、始めに私がなぜテスト駆動開発を必要としたのかの背景を説明したいと思います。皆さんの中で近しい状況があればこの節を読み進めるモチベーションになりますし、逆に読み飛ばす判断材料にもなると思います。（もちろん、是非読んでいただきたい所ですが。。。笑）

事の始まりは業務自動化用のスクリプトを書いている所からでした。以下のような出力された文字列情報を解析して異常がないか確認するためのスクリプトです。

ファイルシステム	サイズ	使用量	残り	...
○○○	80G	5G	74. 3G	
○○○○	42M	20M	21. 5M	
○○○○○○	112M	5M	96. 4M	
○○○○	31M	6M	24M	
○○○○	2. 4G	2. 1G	300M	

図 7.1: 解析対象例

さらっと書けてしまいそうですが、上記の図のような物を判定するときいくつも考えなければならぬことがあります。例えば、

- 入力が想定外の物だったら？
- 予期せぬ内容だったら？
- 条件分岐はあってる？（組み合わせに問題はない？）
- 型変換時ミスはしていない？

とくに Python のように型を明示的に指定しない言語では比較的潜在する想定外のバグが増えやすい傾向にあると思います。もちろん一つずつ抜け漏れなく考えることができれば問題はありません。しかし気をつける事が多いため、実装している間は複雑なイライラ棒をしているような気持ちになりました。さらに弊害として1つの機能を実装するだけでも疲れてしまい、中々思うように開発も進まずフラストレーションのたまる日々でした。

ここまで背景をお話ししましたが、この悩み要約すると

7.2 頭をすっきりテスト駆動開発

- 沢山注意しながら開発をするのが大変

となると思います。この悩みを解決するのがテスト駆動開発でした。

テスト駆動開発のステップ^{*1}は Kent Beck によると

1. まずテストを 1 つ書く
2. すべてのテストを走らせ、すべて成功することを確認する
3. 小さな変更を行う
4. すべてのテストを走らせ、全て成功することを確認する
5. リファクタリングを行なって重複を除去する

とされています。

ここでの大きなポイントはまずテストを書くという事です。小さく機能を開発しテストを実施するというステップを繰り返せば、ミスをした瞬間にテストが指摘してくれます。始めはテストに引っかかるたびに小言を言われているような気がして億劫でしたが、機能の実装を完了してみると思ったよりも時間がかかっておらず疲れも少なかった事を今でも覚えています。これは頭のリソースを機能開発にだけ使う事ができ、バグへの懸念はテストに指摘されたタイミングで修正するだけで良いというのが理由だと考えています。もちろん小さなスクリプトであればテストを最初に書くため書かない場合より時間がかかることは否めません。しかしそのコードが長くなればなるほど複雑さが増すため、頭のリソースを機能開発にだけ向けることができるテスト駆動開発の方が、効率が上がり開発時間が短くなる可能性があります。テスト駆動開発は言わば長距離走向けの手法と言えるでしょう。

あくまで私の体験談でサンプル 1 のお話でしたが、ここまでテスト駆動開発が広まっている事を考えると試す価値はあると思います。この説ではテスト駆動開発の紹介のみにどまりますが興味を持たれた方は、Kent Beck の「テスト駆動開発」もしくは Robert C Martin の Clean Code 第 5 章を参考にしてみてください。

7.2.3 誰が為のテスト駆動？

さてここまで素晴らしい手法としてテスト駆動開発として説明してきましたが、私はテスト駆動開発は常に使うべきとは考えていません。理由としては PJ で開発する以外の場面ではテストを成果としづらいと考えている為です。もちろん、チーム内でテストを残す文化があれば別ですが、チームで成果と見なされないものに工数を掛ける行為は中々継続しないものです。

そもそもの話となります、テスト駆動開発で作成するテストと品質の為のテストでは

^{*1} Kent Beck (2017) 『テスト駆動開発』 和田 隼人訳, P1, オーム社.

第7章 現場で使える Python 自動化入門

目的が異なります。テスト駆動開発で書くテストは開発者の注意点を網羅していれば良く、カバレッジをそこまで意識しなくても問題はないと思われます。品質担保の為の成果物としてテストを残す為にはカバレッジを上げる必要がありますが、そのカバレッジ向上と成果物の品質の向上は対数曲線的な関係にあると考えており幾ら細かいバグをテストしていてもスクリプトの機能自体は向上しません。

趣味ではなく現場で活用することを考えると工数も切り離せないファクターの一つです。成果に対して労力を必要以上にかける事は無駄であると考えて割り切ることは必要だと考えています。逆に言えば、ある程度品質が求められるのであればそれなりの工数が必要となる為、テストする工程を別で考えた方が良いと思います。

以上の事から、私は求められる品質を考慮したうえで開発の為のメソッドの一つとして効率を上げる為だけに利用する事を心掛けています。

7.3 みんなで読もう Sphinx

7.3.1 はじめに

いきなり恐縮ですが、ツールを作成する事の恩恵は自分で使う場合のみにとどまらないと考えています。そしてむしろその多くは他の方に使ってもらう事により得られるものだと思っています。もちろんツールの種別によるとは思いますが、適切に抽象化されたメソッドは他の人に利用、継承してもらう事で複利的にその真価を發揮し続けます。

しかし、このような文化を形成していくためには誰かにその仕様を適切に伝える必要があります。そのためにはドキュメントが必要となるでしょう。皆さんはツールを作成するときにどのようなドキュメントを作成していますか？ README.md を作成する、エクセルで作成する、テキストで作成するなど様々だと思います。

本節ではドキュメントを作成する為に、Sphinx というドキュメントジェネレータを紹介したいと思います。

ドキュメントジェネレータとはその名の通りドキュメントを作成するためのツールです。百聞は一見にしかずという事で、まずは Sphinx の活用事例を見てみましょう。以下は Open3D という点群処理用のライブラリのドキュメント^{*2}です。

見ていただくとわかる通り html 形式の web サイトのようなドキュメントです。

^{*2} <http://www.open3d.org/docs/release/tutorial/geometry/pointcloud.html>

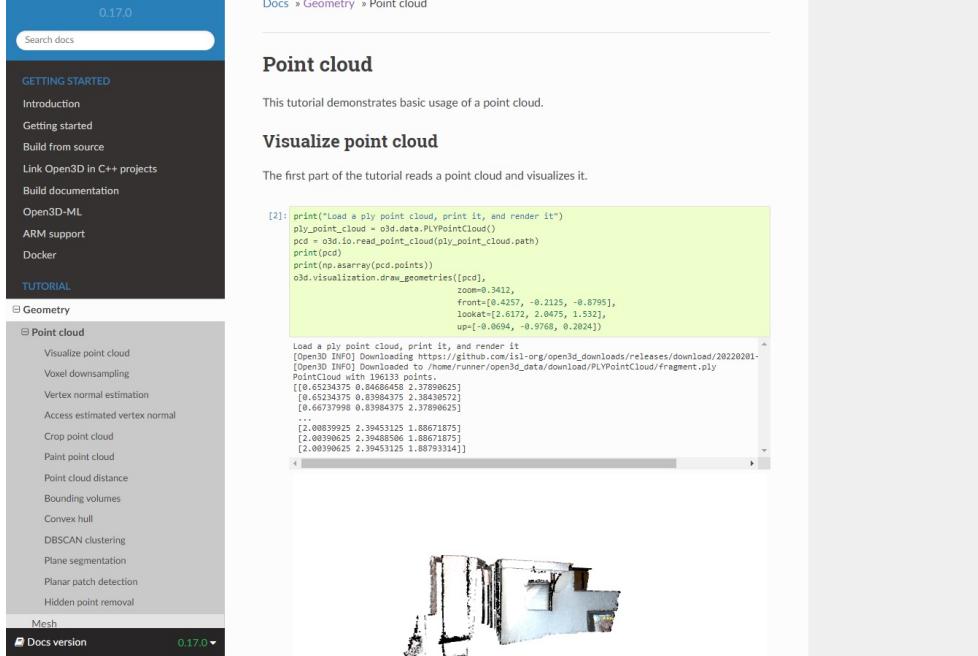


図 7.2: Open3D ドキュメント

一言で言ってしまえば、Sphinx はこの html ドキュメントを作成するためのツールです。（もちろん、html に限らず Latex や ePub, Texinfo, manual pages, plain tex といったフォーマットにも対応しています。^{*3)} 類似のツールとしては Doxygen などが有名です。

Sphinx のようなドキュメント生成ツールでドキュメントを作成した場合のメリットをいくつか列挙してみると、

1. ドキュメントがきれい
2. コメントが残る
3. 検索ができる
4. レイアウトを統一しやすい

といったところでしょうか。

まずドキュメントが綺麗である事は明白です。ただのテキストよりはこちらの方が読みやすいです。次にコメントが残るという点について、詳しくは後述しますが、実はこの Sphinx で生成する為のドキュメントの情報はプログラムにコメントとして残すのです。

^{*3} <https://www.sphinx-doc.org/ja/master/>

第7章 現場で使える Python 自動化入門

またプログラムにコメントとして残されている為に、テキストとして残ることもメリットとなると思います。エンジニアとしてはデータとなるべく環境に依存しない方法で記載をしたいものです。この点においてプログラムのコメントとして残っていれば安心です。

さらに、この Sphinx では検索機能もついてきます。ユーザーフレンドリーな UI があれば他の方にも使ってもらいやすいです。最後にレイアウトを統一しやすいという点も上げられます。いろんな方がテキストでドキュメントを作成するとそのレイアウトは多種多様になりその読みやすさも書き手によってまちまちになります。

最終的にドキュメントの質はその内容に依存しますが、レイアウトが整えられている事で読み手はコンテキストを把握しやすくなるためレイアウトを統一しておく方がベターだと思います。

7.3.2 Sphinx 入門

Sphinx のドキュメント生成方法の基本的なステップは以下のとおりです。

1. コメントを記載する
2. rst ファイルを生成する
3. ドキュメントを生成する

ここからは、実際に Sphinx でドキュメントを生成する流れをご紹介します。

Sphinx はまずコメントを書きます。今回は Google の Docstring 形式^{*4}で記述します。このほか、numpy 形式などもあるので好みのスタイルを使いましょう。

< test.py(Google Docstring 形式コメント入り) >

```
class test:  
    """  
    testメソッド  
    """  
    def __init__(self, n):  
        self.number = n  
  
    def add(self, a):  
        """  
        内部変数nにaを足す関数  
  
        Args:  
            a(int): 内部変数nにaを足す  
  
        Returns:  
    """
```

^{*4} <https://google.github.io/styleguide/pyguide.html> 3.8.2 Modules

7.3 みんなで読もう Sphinx

```
int : 足し算後の内部変数
"""
self.number += a
return self.number

def out(self):
    """
    内部変数を出力するだけの関数

    Returns:
        None
    """
    print(self.number)
```

次にコメントから reStructuredText (rst ファイル) を作成後、ドキュメントを生成します。環境依存のある config の記述など詳細部分は省きましたが、ほぼこれだけで以下のようなドキュメントが生成されます。



図 7.3: ドキュメント例

また、Sphinx では html テーマがいくつもある為、自分好みに様々なカスタマイズを行なう事ができます。以下でいくつかのテーマを紹介いたします。(個人的には Open3D でも使用されているサードパーティー製の Read the Docs が好みです。)

第7章 現場で使える Python 自動化入門



図 7.4: classic

Welcome to test's documentation!



図 7.5: Alabaster

View page source

Welcome to test's documentation!

Contents:

- [test module](#)

Indices and tables

- [Index](#)
- [Module Index](#)
- [Search Page](#)

Next

© Copyright 2023, Nishino.

Built with Sphinx using a theme provided by Read the Docs.

図 7.6: Read the Docs

7.3 みんなで読もう Sphinx

7.3.3 型アノテーションのすすめ

さて少し Sphinx とは離れますか、コードに仕様を記載するという点で型アノテーションについてご紹介したいと思います。型アノテーションは Python3.5 から追加された機能で Python の引数の型と返り値の型を記述する方法です。あくまでもアノテーションの為 Python 自体の挙動には関係がありません。以下にアノテーションの例を示します。

例えば add メソッドには引数の隣に int 型と記載してあり、返り値の型は” -> ”で記述されています。詳しくは Python.org のドキュメントを参照ください。^{*5}

```
< test.py >
```

```
class test:
    def __init__(self, n:int):
        self.number = n

    def add(self, a:int) -> int:
        self.number += a
        return self.number

    def out(self) -> None:
        print(self.number)
```

型アノテーションは Python で定義されている為、通常のコメントとは異なり IDE などを使うと自動で読み取ってくれる場合もあります。例えば Visual Studio Code で add 関数を呼び出すと以下のように型ヒントを自動で示してくれます。

^{*5} <https://docs.python.org/ja/3/library/typing.html>

第7章 現場で使える Python 自動化入門

```
1  from test import test
2
3  def main():
4      n=10
5      test_instance = test()
6      test_instance.out((a: int) -> int)
7      test_instance.add()
8      test_instance.out()
9  if __name__ == '__main__':
10     main()
```

図 7.7: Visual Studio Code による型ヒント

その他サードパーティーライブリの mypy^{*6}などを使用すると、型ヒントから型チェックを行なう事ができます。例えば以下のように、int で定義した変数に対して文字列を代入するようなコードを書いたとします。ここで実行すると後から代入した文字列が表示されます。

しかし、mypy を実行すると以下のようにエラーが出力されます。エラー文を読むと 2 行目で定義されている型と一致しないという事のようです。C 言語等のコンパイルチェックのようですね。これにより事前の型チェックを行なう事ができるようになり、型誤りによるバグの混入を未然に防ぐことができます。

< mypy_usecase.py >

```
def main():
    a: int = 3
    a = "bug"
    print(a)

if __name__ == '__main__':
    main()
```

<実行結果>

^{*6} <https://github.com/python/mypy>

7.4 最後の救い log 取得

bug

<mypy 実行結果>

```
(huga)hoge@hogehoge $ mypy mypy_usecase.py
mypy_usecase.py:2: error: Incompatible types in assignment (expr→
expression has type "str", variable has type "int") [assignment]
Found 1 error in 1 file (checked 1 source file)
```

7.3.4 現場での活用方法

現場で使用する場合には、生成されたドキュメント群を共有ディレクトリに置いています。エクスプローラーでパスがたどれるのであればドキュメント用に新規でサーバーを建てずともブラウザから確認することが可能です。

7.4 最後の救い log 取得

7.4.1 はじめに

みなさんは log を残しているでしょうか？先ほどの「2.1 はじめに」で述べたように私はなるべく多くの方に使ってもらう事でツールは真価を発揮し続けるという考えをもっています。沢山の方が使えば使うほど想定外の問題が起きる可能性は上がります。仕様による動作から想定しないバグまでできる事であれば全て網羅したいところですが、ツール作成時には必要以上の仕様を入れ込むことはコストパフォーマンスの低下を招きます。また、必要でない機能は仕様を決める時の解像度が低い為、良い仕様にすることは難しいものです。

そこでツール使用者に適切な情報を与えるために log 機能が必要になってきます。本節では、ツールの使用者に適切な情報を与えるための log についてお話ししたいと思います。

7.4.2 log を設定しよう

まず、前提としてどのような log が必要となるでしょうか？これは目的によって様々となります。例えばレポートログを作成するツールであれば形式は人が読みやすい書式が

第7章 現場で使える Python 自動化入門

良いでしょう。今回はツールのデバッグ用のログとして考えてみましょう。

ログにはロギングレベルというものがあります。例えば Python の logging モジュールでは以下の 6 つが定義されています。^{*7}

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG
- NOTSET

なぜこんなにも分れているかというと、常に出してほしい情報と必要な時に出してほしい情報のレベルは異なる為です。

例えば、セキュリティソフトをイメージしてみましょう。デバッグ用に必要な情報はなるべく多く出したいのですが、常に今どのファイルを検査していて、どのようなアップデートファイルをインストールしていて、バックグラウンドでどのシステムが動いているといった通知が常に出ていると必要な情報が埋もれてしまい逆に判りづらくなってしまいます。

一方で、攻撃を受けた場合は即座に通知をしてほしいものです。このように、その必要性に応じて適切にロギングレベルを適切に管理する必要があります。これだけの種類が用意されているとはいえ私が現場で使う際は CRITICAL、WARNING、DEBUG 程度でしか使い分けをしていません。理由としては Python で作成するツールの多くはそこまで規模が大きくならない為です。粒度は必要に応じて定めてよいと思います。

Python であれば、logging モジュールで必要なレベルを設定しておけば import 先で logging モジュールを使っていれば log に情報が吐き出されます。適切に設定しておくことでユーザーフレンドリーなツールとなるでしょう。また使用者側で確認ができればツールから手を離すことができます。現場ではツール開発者がメンテナンスを行なうことが多いと思います。ログを残してなるべく使用者側の手で解決してもらえるようにしておくと良いと思います。

7.4.3 DEBUG は DEBUG

DEBUG ログはバグ調査の最後の手段としてあるべきだと考えています。そのため、なるべく DEBUG では様々な情報があったほうが良いのですが、それでも必要な情報は精査すべきです。

^{*7} <https://docs.python.org/ja/3/library/logging.html?highlight=log>

7.5 さいごに

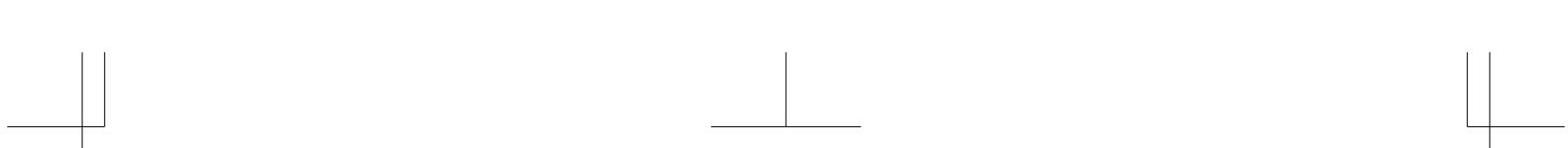
これは、私自身の体験なのですが何度も呼び出されるモジュールに対して細かく DEBUG を吐き出すように設定していました。何度かテストを行なってログを確認しようとするとテキストエディタでは開けなくなってしまいました。これは、あまりにもログを細かく設定しすぎたために一度の呼び出しで沢山のログが吐き出され（一度で約 500MB 程度）いつの間にかログが肥大化していたためです。DEBUG の設定はデバッグの為に必要な情報を精査すべきでした。

この例は、あくまでも私の場合であり状況に応じてログを残すべきですが、サーバー上に置くものであればログがディスクの圧迫を招く可能性があるという事も考慮に入るべきでしょう。また事前にログの量をある程度見積もることも重要なと思います。

7.5 さいごに

本章の裏テーマは「開発者の負担を減らす」というものでした。Python では様々なモジュールがでており、また書籍やテックブログも潤沢である事でツールを開発しやすくなっていると思います。

一方で現場ではその多くが開発者自身で運用やメンテナンスにも時間をさく必要があり、ツールを開発すればするだけメンテナンスコストが高まるという負のサイクルに課題を感じています。そのため本章の 2/3 がツール利用者に自力で解決してもらう為の同線を敷く為の Tips となっています。ご紹介した内容が少しでも有益なものとなれば幸いです。本章をお読みいただきありがとうございました。



第8章

GAS と AWS を使って Web 操作 をハックしてみた!

大野 泰歩

8.1 はじめに

はじめまして！ ヤスと申します。

私は普段通信事業者として商用設備の運用保守業務を担当しております。運用保守業務を実施されたことのある方であれば経験があるかと思いますが、保守しているシステムの不具合や故障の対処を日々実施していく必要があります。この業務を滞りなく遂行していくには情報の扱いを出来る限り自動にしていく事が望ましいです。

私は Google Apps Script^{*1}(GAS) や Amazon Web Services^{*2}(AWS) を用いて Web ページの操作を自動化の部分を実施しましたが、こちらをご説明する前に既存で作成されていた情報収集の自動化部分からも記載していこうかと思います。

GAS や AWS などの技術については独学で学んだモノになりますので、拙い部分も多々あるかと思いますが、あくまで自動化の一例として温かい目で拝読頂けると幸いです。

8.2 情報収集の自動化 (GAS)

「はじめに」で少し触れたように、私は運用保守業務を実施するうえで様々なインシデントを管理しており、こちらを管理するのにあたりスプレッドシートを用いております。毎度手動で入力していくのは大変なため、Gmail で通知されるインシデントから必要な情

^{*1} <https://developers.google.com/apps-script?hl=ja>

^{*2} https://docs.aws.amazon.com/ja_jp/

第8章 GAS と AWS を使って Web 操作をハックしてみた!

報を抜き出し、スプレッドシートに転記されるように GAS を用いて自動化しています。

自動化の方法としては通知されるインシデントメールのテンプレートを定義することで GAS を用いて必要項目を抜き出しひスプレッドシートに転記するというものになります。

メールの例文を 2 種類程紹介します。

- ・メールサンプル 1

```
■ID  
1111111  
  
■ホストグループ名  
service_ABC  
  
■ホスト名  
hostname_ABC  
  
■アラームテキスト  
• Error message
```

- ・メールサンプル 2

```
[Linuxtime] 1111111  
[Hostname] hostname_ABC  
[Serial] abc123
```

ご紹介した様にテンプレートとしてメール本文の内容を決めることが出来た場合は、 GAS を用いてメール本文から情報を抜き出し、スプレッドシートに転記することが出来ます。

今回は正規表現を用いて情報を抜き出す際の例を記載いたします。

- ・メール本文の抜き出し (GAS)

```
var myThreads = GmailApp.search('メール件名', 0, 50);  
var myMsgs = GmailApp.getMessagesForThreads(myThreads);  
  
for ( var threadIndex = 0 ; threadIndex < myThreads.length ; threadIndex++ ) {  
    var mailBody = myMsgs[threadIndex][0].getPlainBody();
```

8.3 Web 操作の自動化 (AWS)

- メールサンプル 1_抜き出し方法例 (GAS)

```
var myRegexp = new RegExp('■ID' + '[\\s\\S]*?' + '■');
if (mailBody.match(myRegexp)) {
    var incidentID = mailBody.match(myRegexp)[0].split('\\n')[→
1];
}
else {
    continue;
}
```

- メールサンプル 2_抜き出し方法例 (GAS)

```
var myRegexp = new RegExp('\\[Linuxtime\\]' + '.*?' + '\\r');
if (mailBody.match(myRegexp)) {
    var incidentID = mailBody.match(myRegexp)[0].replace('[Linux→
time]', '');
}
else {
    incidentID = "Linuxtime未記載";
}
```

このようにメール本文から情報を変数として取得することができましたので、後はスプレッドシートの中で転記したい位置に取得した情報を記載するようにすれば情報の取得について自動化することが出来ました。

その他に補足情報が必要な場合は、他のスプレッドシートに記載している情報を同じ様に GAS を用いたり関数を使用することで更に多くの情報を集めることもできます。

8.3 Web 操作の自動化 (AWS)

情報の取得について自動にすることが出来たので、集めた情報を元にどの様に Web 操作を自動にしていくかを記載いたします。

元々は GAS を用いて Web 操作を実施しようと思っておりましたが、GAS のみで Web 操作を可能とするドキュメントを見つけることが出来なかったため、AWS を用いて自動化していく事にしました。

またこの後説明していく処理の流れがイメージしやすいように概要を添付いたします。

第8章 GAS と AWS を使って Web 操作をハックしてみた!

概要図

GASで必要な情報を「API Gateway」に渡し、
その後「Lambda」で処理の実行（pythonでselenium）

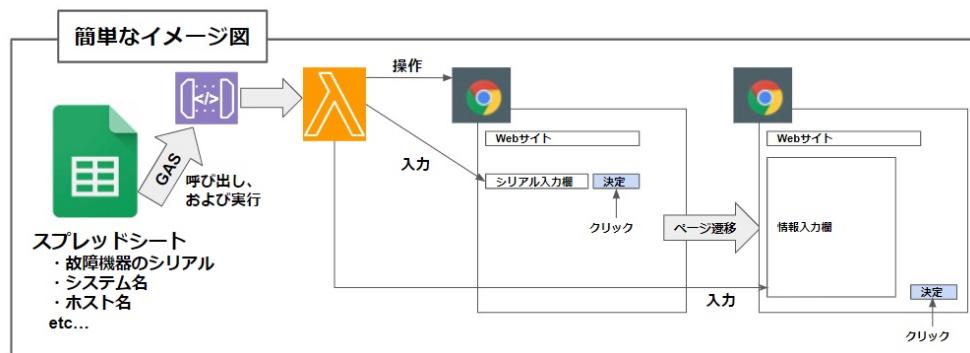


図 8.1: 処理概要

添付した「処理概要」の通りになりますが、下記の流れで処理を行っていきます。

1. スプレッドシートに情報収集
2. GAS から API Gateway に Lambda の呼び出し処理
3. Lambda にて Web 画面の自動操作
4. 処理結果を GAS に返信

まずは Lambda での処理を記載していきます。今回 Lambda では Python を使用していきます。Python を選択した理由としては Web 画面の操作を実施するのに有名な Selenium というモジュールを使用するためとなります。

自動化をするにあたって、このタイミングで 1 度躊躇しました。躊躇した理由としては Lambda の中には標準的な関数しか搭載されておらず Selenium などの拡張モジュールを使用することが出来なかったからとなります。そのため拡張モジュールを使用するために Lambda の中のレイヤーという機能として使用したいモジュールの追加を行いツールを実行出来るようにしました。

今回追加したモジュールは 4 つになります。

- Chromedriver: Selenium の処理を行う土台のサイトとして使用するため
- headless-chromium: Web 画面を表示しない状態で使用するため
- Selenium: Web 画面操作の自動化を行うため
- headless-selenium: ヘッドレスモードで Web 画面操作を行うため

8.3 Web 操作の自動化 (AWS)

レイヤー (4)				最終取得済 26 分前	C	レイヤーの作成
名前 バージョン 互換性のあるランタイム				互換性のあるアーキテクチャ		
chromedriver	2	python3.7	-			
headless	1	python3.7	-			
headless-chromium	2	python3.7	-			
selenium	2	python3.7	-			

図 8.2: レイヤー情報

ここまで準備出来たら後は python を用いて Selenium の処理を記載することで Web 画面の操作が出来るようになります。

- import 内容 (Lambda)

```
import json
import time
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.select import Select
```

- API Gateway と連携するための関数 (Lambda)

```
def lambda_handler(event, context):
    # TODO implement
    body = json.loads(event['body'])
    sample_value = sample_function(body)

    return {
        'statusCode': 200,
        'body': json.dumps(sample_value)
    }
```

- Chrome ドライバの設定 (Lambda)

第8章 GAS と AWS を使って Web 操作をハックしてみた!

```
def headless_chrome():
    ## Seleniumが使用するWebDriver作成時のオプションを作る
    options = webdriver.ChromeOptions()
    ## オプションのバイナリロケーションにLayerで用意したheadless-chromiumのパスを指定
    options.binary_location = '/opt/headless/bin/headless-chromium'
    ## オプションにヘッドレスモードで実行させる記述を書く
    options.add_argument("--headless")
    options.add_argument("--no-sandbox")
    options.add_argument("--single-process")
    options.add_argument("--disable-gpu")
    options.add_argument("--homedir=/tmp")

    driver = webdriver.Chrome(
        ## Layerで用意したchromedriverのパスを指定
        executable_path="/opt/headless/bin/chromedriver",
        options = options
    )
    return driver
```

- Selenium 使用例 (Lambda)

```
def sample_function(body):
    driver = headless_chrome()
    driver.delete_all_cookies()
    driver.get('https://www.google.co.jp/')
    search_box = driver.find_element_by_xpath('//*[@id="APjFqb"]')
    search_box.send_keys(body['sample'])
```

この4つの内容を記載することでプログラムの基本的な概要は完成しております。後は、実際に使用する場合に合わせて必要なオプションを追加したり Selenium の処理を追記していく事で実行したい処理を完成させていく事が出来ます。

Selenium の使い方や関数の詳細については色々と書籍や Web ページでの説明があるので、今回の自動化をする際に良く使用していた関数を代表として紹介します。

- find_element_by_xpath: Web ページの xpath を指定して要素を取得する
- send_keys: テキストボックスなどに文字列を入力する
- select_by_value: ドロップダウンなどのセレクト要素を選択する
- click: 選択している要素をクリックする

8.4 GAS から Lambda の呼び出し (GAS)

これにて Web 画面の操作を自動にする処理を Lambda を用いて作成することが出来ました。そのため後は API Gateway を作成し、今回作った Lambda と連携させることで GAS から呼び出せるようになります。

8.4 GAS から Lambda の呼び出し (GAS)

GAS から Lambda に連携するにあたって API Gateway で生成した連携用の URL と収集した情報を json の形で整えて使用する必要があります。その整えた情報を UrlFetchApp.fetch を使用することで Lambda に渡すことが出来ます。

- GAS から Lambda への連携 (GAS)

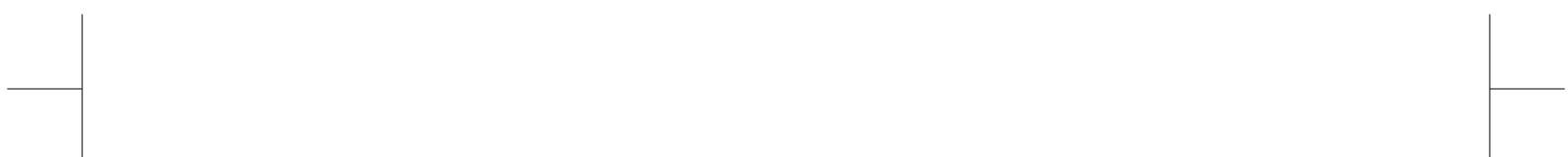
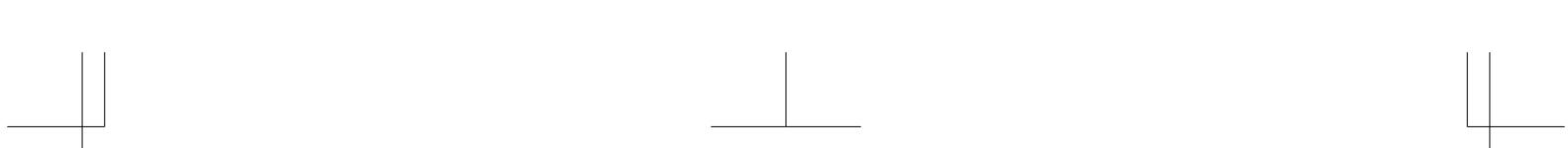
```
const aws_url = 'API GatewayのURL';
const params = {
  'method' : 'post', //get or post
  'contentType': 'application/json',
  'payload' : JSON.stringify(
    {"sample_value1" : sample_value1,
     "sample_value2" : sample_value2,
     "sample_value3" : sample_value3})
};

const req = UrlFetchApp.fetch(aws_url, params);
```

これにて GAS と Lambda の連携が完了しましたので、後は処理が完了した旨のポップアップを表示させたりメールを送るなどお好みで設定するかたちとなります。

8.5 さいごに

本章においては、定常的な業務の効率化を行うための方法について記載させて頂きました。AWS の使い方や GAS の使用方法について他にも最適な方法はあるかと思いますが、この章をお読みいただいた方の業務が少しでも楽になったり、効率化への閃きに寄与することができますれば幸いです。最後になりますが本章をお読みいただきましてありがとうございました。



第9章

フローで思い通りの結果を得るために大切なこと

百合宮 桜

9.1 はじめに

みなさま、ごきげんよう。突然ですが、Power Automate でフローを組んで、テストして、無事に成功した時、安心してそのままウィンドウを閉じていませんか？「フローが成功しました」という表示は「エラーなしで正しく実行されました」という意味で、「作成者であるあなたの想定通りに実行されました」という意味ではありませんよ？

さて、あなたが今閉じてしまったそのフロー、本当に「想定通り」の動きをしているでしょうか？不安になっていませんか？本章では、フローが「想定通り」に動いているかどうかの確認方法を実際のユースケースを見ながら、解説していきます。ぜひ最後まで一読頂き、想定通りの処理が行われているかどうかの確認方法を身に着けて頂きたいと思います。フローの処理に不備があったがために、偉い人に「手作業憐みの令」を出されないように。

9.2 テストは、2段階に分けられる

IT を本職とされている人よりも知識が薄い市民開発者が忘れがちなのは、テストは次の2段階に分けられるということです。

1. テスト機能を使用したフローの動作テスト
 2. 作成者が期待する「正常な動作」が行われているかの確認
1. は、ほとんどの人が当たり前のように行っていることだと思います。問題は 2。作成者

第9章 フローで思い通りの結果を得るために大切なこと

が期待する「正常な動作」が行われているかの確認です。これは、フローのテスト結果の中身を確認したり、フローの外に出力された結果 (EX: フローで投稿した Teams のメッセージ) を確認したりすることが必要です。

フローが成功していても、作成者が期待する「正常な動作」をしていないこともよくあります。市民開発者は、Power Platform での開発が初めての開発経験という方も多く、1. のテスト機能を使用したフローの動作テストが成功すると安心してテストをやめてしまうこともあるのです (かつての私がそうでした) そのため、実際のユースケースを通して、2. の確認方法を学んでいきたいと思います。

9.3 Case アップロードしたはずのファイルが開かない

9.3 Case アップロードしたはずのファイルが開かない

9.3.1 どんなフローか

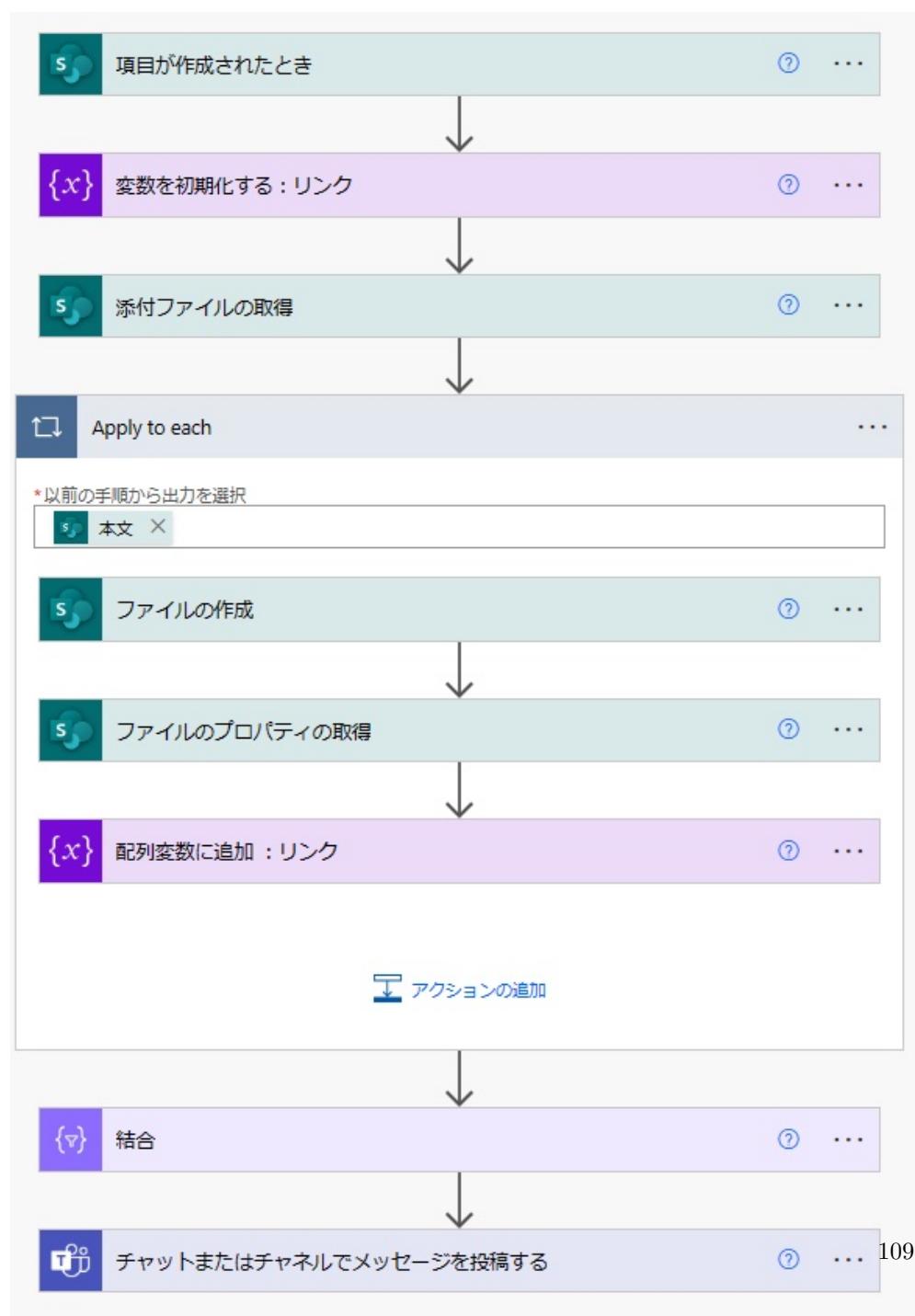


図 9.1: どんなフローか

第9章 フローで思い通りの結果を得るために大切なこと

このフローは、指定された SharePoint リストにレコードが作成された時にそのレコードについている添付ファイルを取得し、その添付ファイルを指定されたドキュメントライブラリにアップロードし、Teams にドキュメントライブラリへのリンク付きメッセージを投稿するという動作をします。

9.3.2 このフローで期待される正常な動作はどういうものか

- 新しいレコードが作成された時にフローが実行されること



図 9.2: 新しいレコードが作成された時にフローが実行される

- Teams にファイルのリンク付きメッセージが投稿されること

9.3 Case アップロードしたはずのファイルが開かない

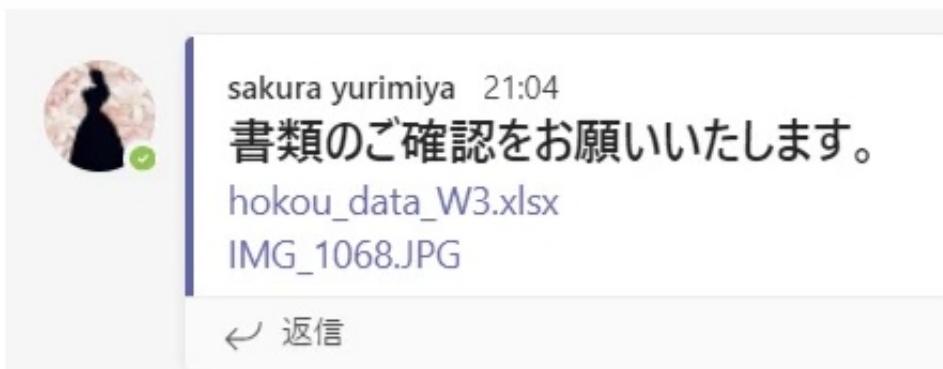


図 9.3: Teams にファイルのリンク付きメッセージが投稿

- メッセージのリンクをクリックしたら、ファイルが表示されること

以上 3 点が作成者である私が期待する正常な動作です。

9.3.3 テスト機能を使って、フローの動作テストを行う

フローを作成したら、まずはテスト機能を使って、動作テストを行います。動作テストは、画面右上の「テスト」ボタンをクリックすると行うことができます。

第9章 フローで思い通りの結果を得るために大切なこと

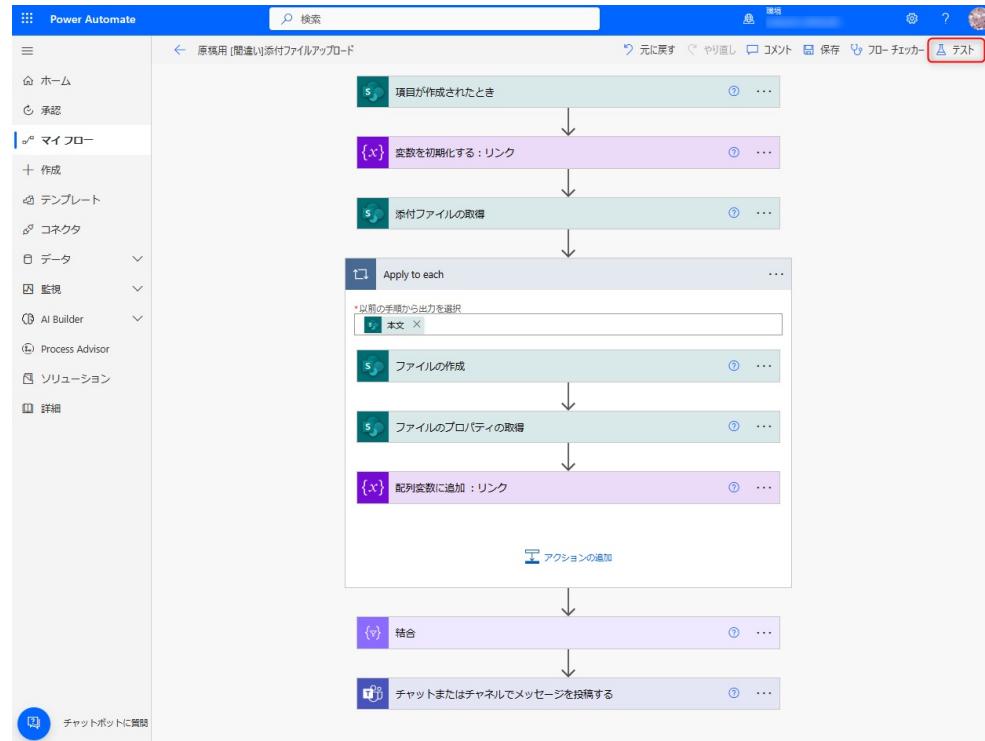


図 9.4: テスト機能 1

9.3 Case アップロードしたはずのファイルが開かない

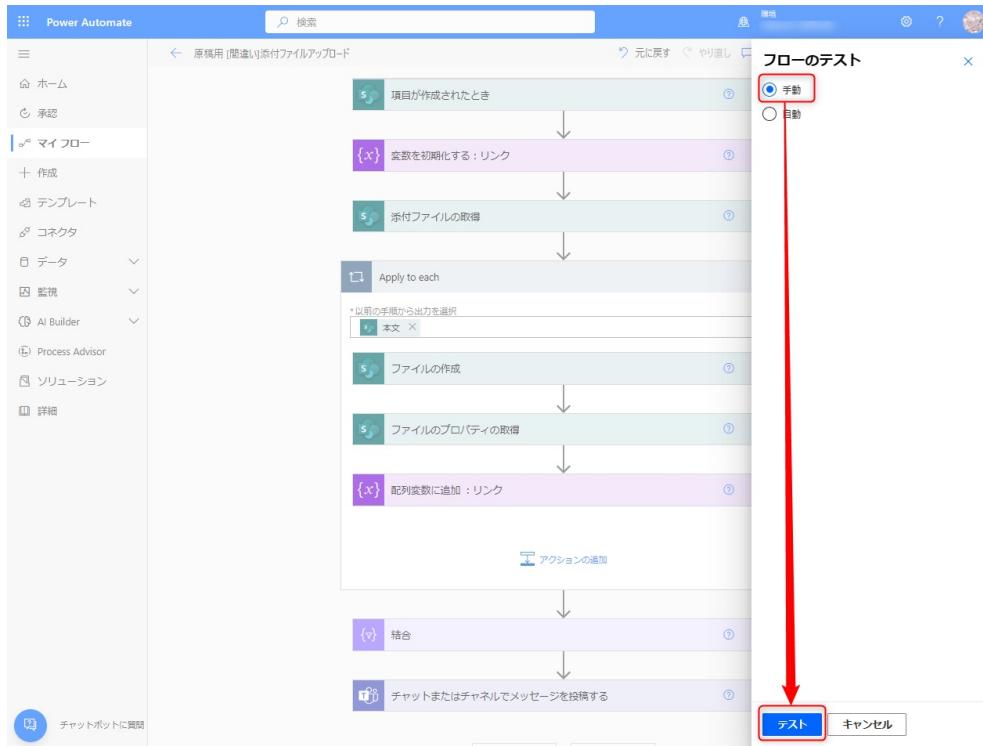


図 9.5: テスト機能 2

最初のテストは「手動」から行います。今回のトリガーは、SharePoint リスト のレコードの新規作成なので、テストボタンを押した後にレコードの新規作成を行ってください。

第9章 フローで思い通りの結果を得るために大切なこと

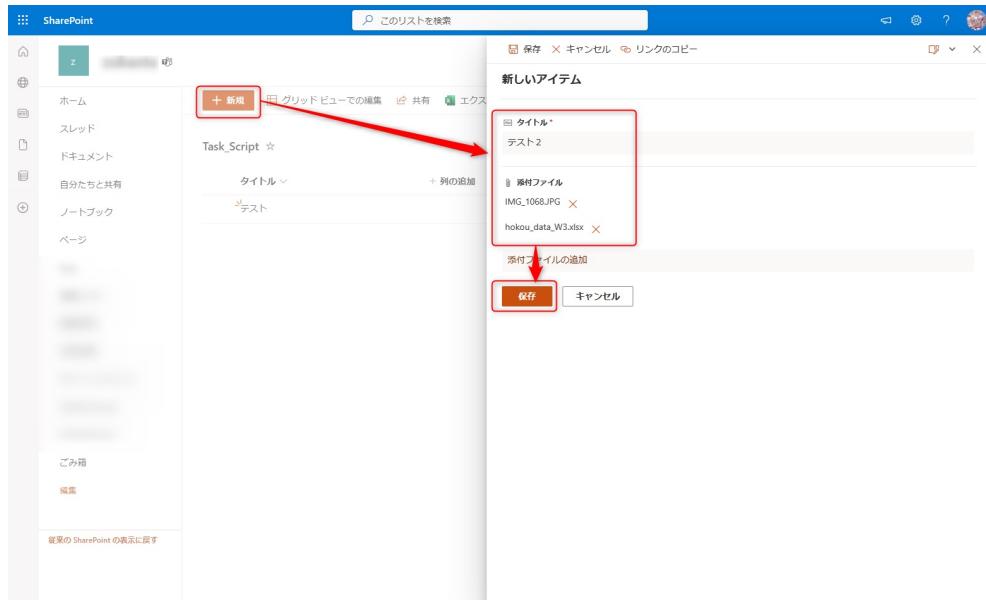


図 9.6: SharePoint のレコードの新規作成

そうすることでフローが実行されます。テストが成功すると、緑色で「ご利用のフローが正常に実行されました」という通知が出てきます。

9.3 Case アップロードしたはずのファイルが開かない



図 9.7: フローの実行

ちなみに 2 回目以降は、1 回目のデータを利用して、「自動」からテストを行うことも可能です。

第9章 フローで思い通りの結果を得るために大切なこと

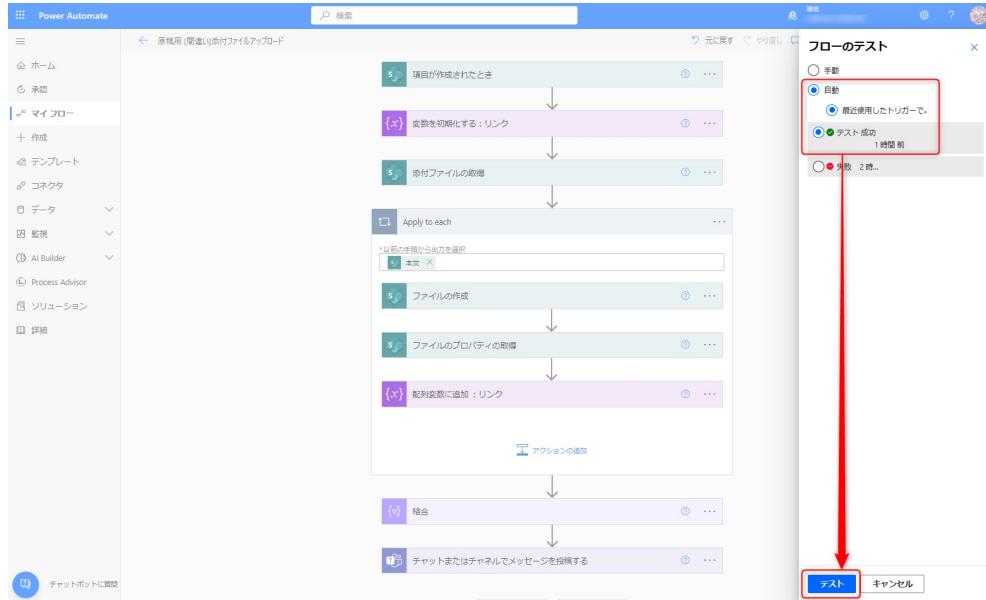


図 9.8: 自動からテストを実行

9.3.4 出力結果が正常な動作をするか確認する

フローが成功し、無事 Teams にメッセージが投稿されたので、リンクをクリックして、ファイルを確認してみます。

9.3 Case アップロードしたはずのファイルが開かない

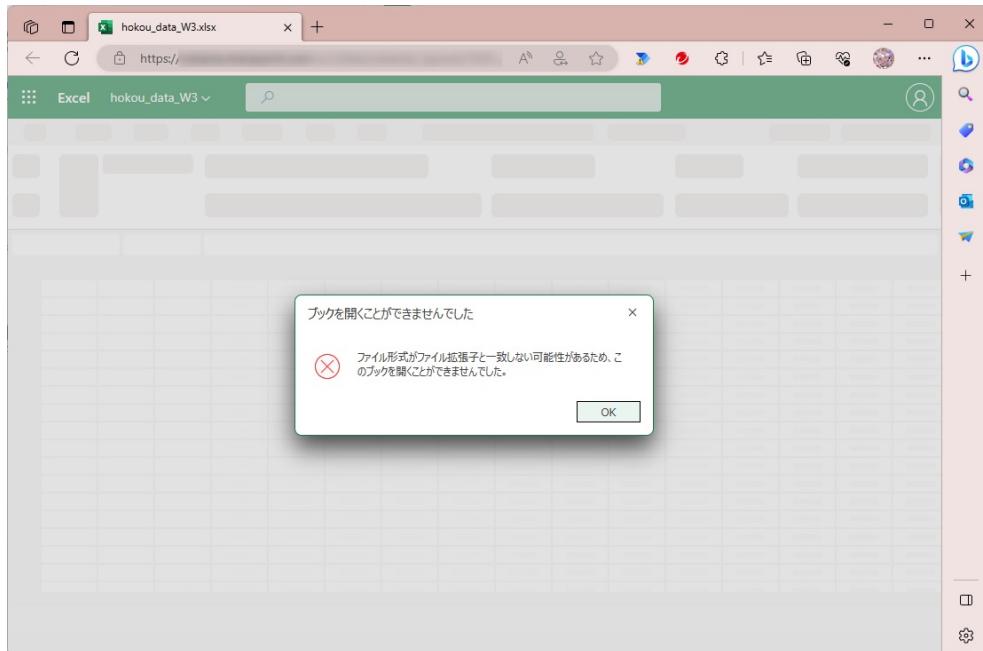


図 9.9: 出力結果

すると、なんということでしょう！！「ブックを開くことができませんでした」というメッセージが表示されています。これは Excel 固有の現象なのか、それとも画像など他のファイルでも同じようになるのか調べるために、画像のリンクもクリックしてみます。

第9章 フローで思い通りの結果を得るために大切なこと

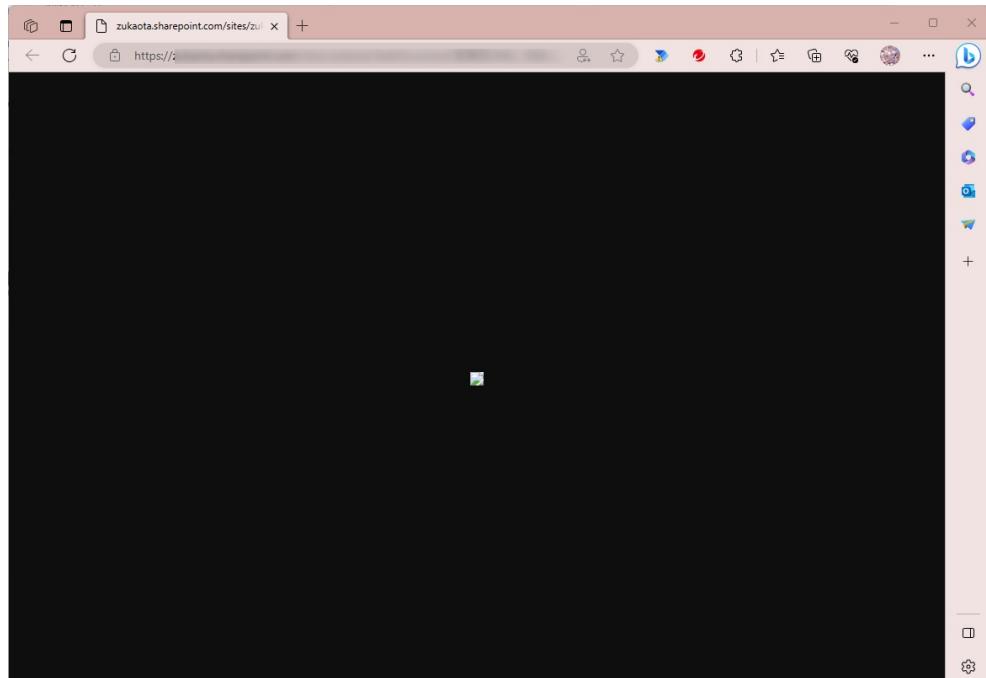


図 9.10: 画像の表示

すると、画像も開くことができませんでした。ものの見事に真っ暗です。つまり、「なんらかの事情でメッセージにあるリンクが開けない」というバグが見つかりました。

9.3.5 「心当たり」をすべて確認し、問題を切り分ける

バグの処理は、無くしたものを探すことと似ている気がします。例えば財布を落とした時、どうするでしょう？ 最後にどこで使ったかを考え、最後に使った場所から現在地までどのように行動したかを思い出して、財布を落としたかも？ と心当たりのある場所を1か所ずつ訪れて、探していきませんか？ バグの処理も同じです。「心当たり」を1か所ずつ確認していきます。私は、2つの「心当たり」があります。

- 実はフローが失敗している（テスト結果は見間違いだった）
- 添付ファイルがドキュメントライブラリにアップロードされる過程で壊れた

9.3 Case アップロードしたはずのファイルが開かない

9.3.6 実行履歴を確認してみる

フローのテストを実行し、成功通知を確認しているとはいえ、バグが出ている以上は本当に成功しているのかの確認は必要です。実行履歴を確認してみましょう。フローの編集画面左上の「←」ボタンをクリックして、1つ前の画面に戻ります。



図 9.11: 1つ前の画面に戻る

「28日間の実行履歴」から先ほどのテストの日時をクリックしましょう。

第9章 フローで思い通りの結果を得るために大切なこと

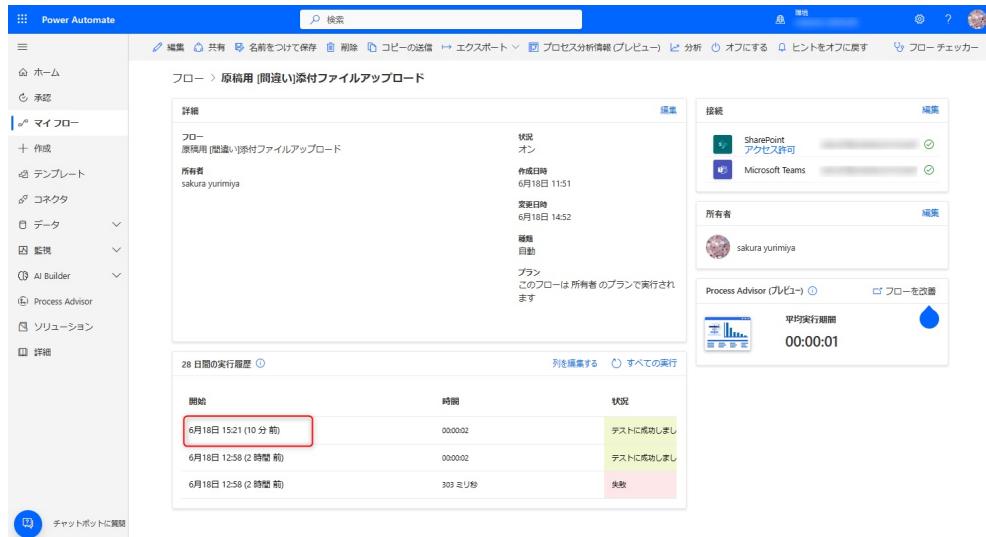


図 9.12: 28 日間の実行履歴

すべてのアクションに緑のチェックマークがついていて、画面上部に成功通知も出ています。間違いなく成功していることが確認できました。

9.3 Case アップロードしたはずのファイルが開かない



図 9.13: 正常に終了

9.3.7 ファイルサイズを確認してみる

フローが正しく動作していることがわかりました。それならば、なんらかの原因でファイルが壊れている説が有力になってきますね。添付ファイルと同じものがドキュメントライブラリに正しく保存されていれば、ファイルの大きさも全く同じなはず……。なので、元データとドキュメントライブラリ保存のファイルサイズを比較してみることにします。

第9章 フローで思い通りの結果を得るために大切なこと

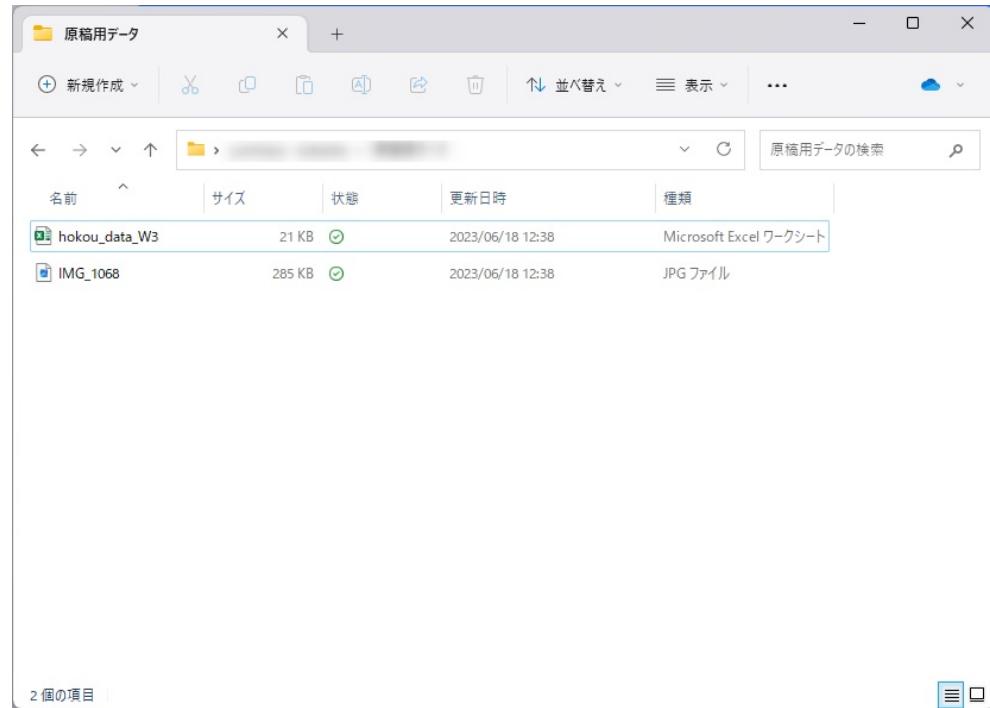


図 9.14: 元データ

TaskDocument > 営業部

名前	ファイル サイズ
hokou_data_W3.xlsx	71 バイト
IMG_1068.JPG	65 バイト

図 9.15: ドキュメントライブラリのデータ

比較してみると、明らかにサイズが異なります。元データは単位が「KB(キロバイト)」

9.3 Case アップロードしたはずのファイルが開かない

なのに対し、ドキュメントライブラリ側は「B(バイト)」です。元データよりもファイルが小さすぎることがわかりました。ここで思い出してほしいのは、ドキュメントライブラリのファイルは Power Automate のフローで作成しているということです。つまり、フローの動作テストは成功しているが、フローに何か不具合があるためにファイルが正しく作成されておらず、ファイルサイズが元データと異なる、という状態であることがわかります。

9.3.8 ファイルの作成アクションを確認してみる

フローの中でドキュメントライブラリに保存するファイルを作成しているのは「ファイルの作成」アクションです。このアクションの設定が正しいかを確認していきます。編集画面から見ると、ファイルの作成アクションの設定は以下のようになっています。

第9章 フローで思い通りの結果を得るために大切なこと

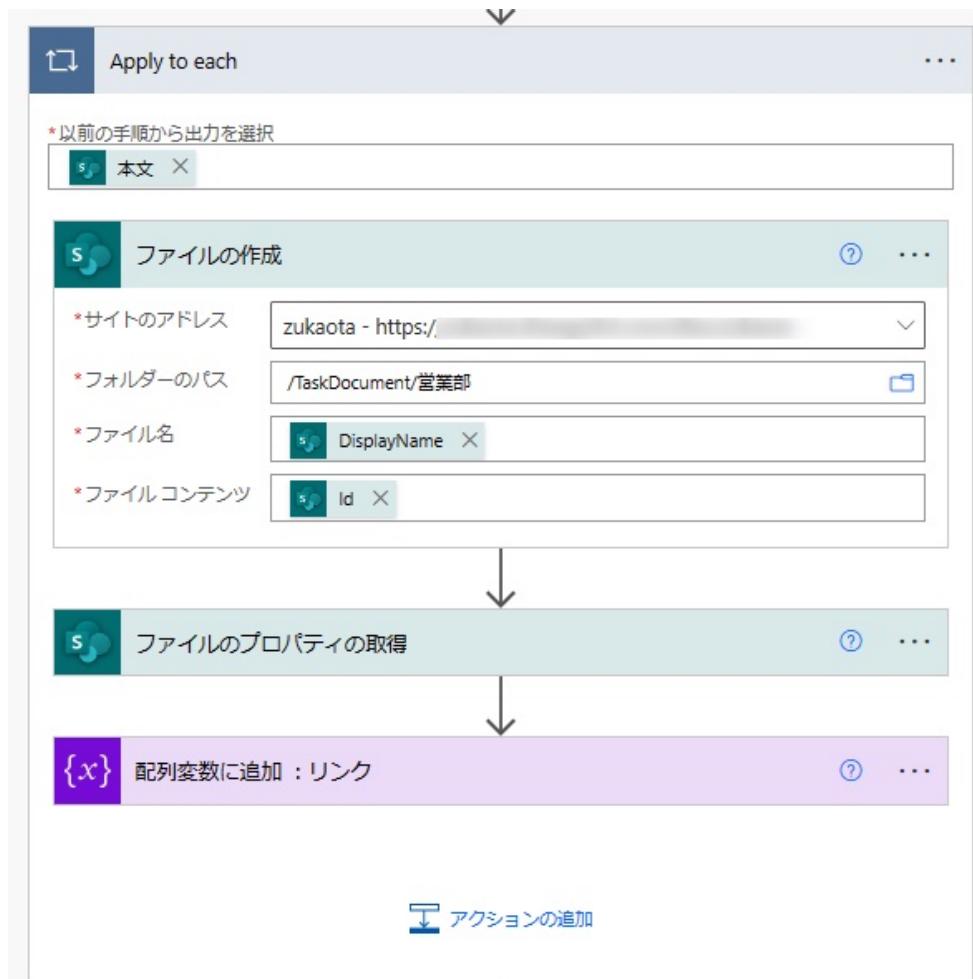


図 9.16: ファイルの作成 1

実行履歴からテスト時に入力された値を確認していきます。

9.3 Case アップロードしたはずのファイルが開かない

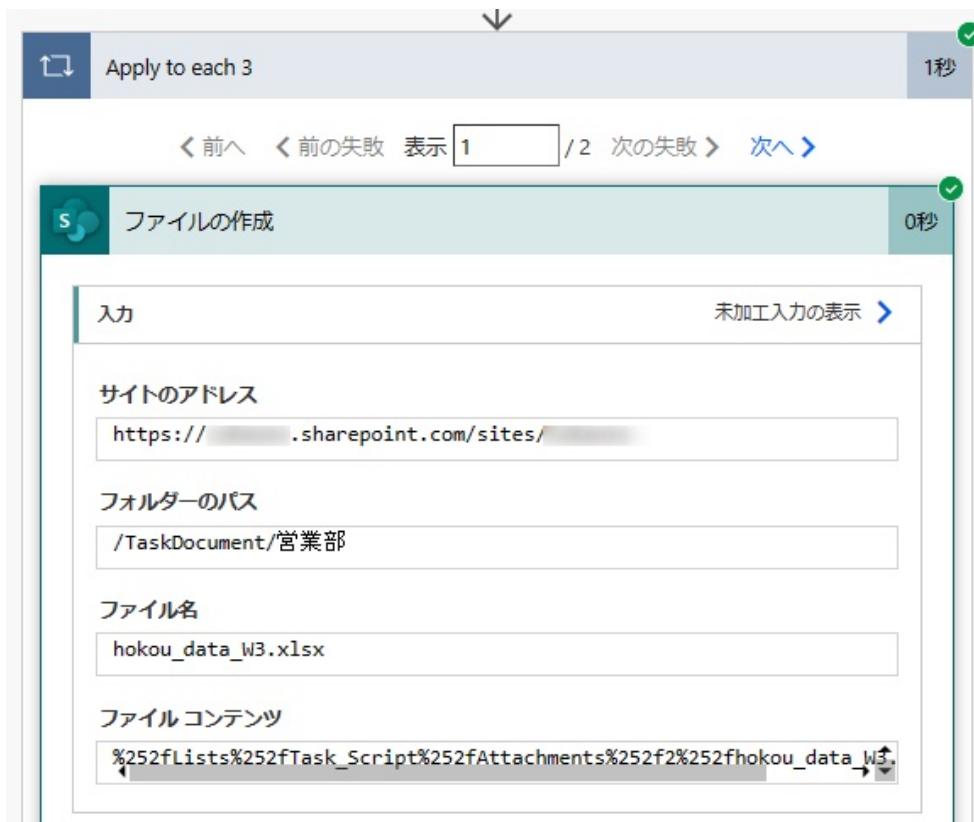


図 9.17: ファイルの作成 2

サイトのアドレス、フォルダーのパス、ファイル名は想定通りの値が入力されていることが確認できます。この 3 つは問題ないでしょう。でもファイルコンテンツは、不思議な並びの文字列で、この値が正しいものなのかどうか、現時点ではわかりません。改めてファイルの作成アクションとは何かを調べてみる必要がありそうです。

言葉の意味を調べる時は公式リファレンスをあたるのが 1 番です。今回の場合は、Power Automate のフローなので、Microsoft Learn^{*1} を読み解いていきます。

*1 <https://learn.microsoft.com/ja-jp/connectors/sharepointonline/>

第9章 フローで思い通りの結果を得るために大切なこと

ファイルの作成

操作 ID: CreateFile

SharePoint サイトにファイルをアップロードします。必ず既存のライブラリを選択してください。

パラメーター

Name	キー	必須	型	説明
サイトのアドレス	dataset	True	string	例: https://contoso.sharepoint.com/sites/sitename 。
フォルダのパス	folderPath	True	string	既存のライブラリから開始する必要があります。必要に応じてフォルダを追加します。
ファイル名	name	True	string	ファイルの名前。
ファイルコンテンツ	body	True	binary	ファイルのコンテンツ。

図 9.18: ファイルの作成 (出典:Microsoft Learn)

Microsoft Learn には、ファイルのコンテンツ欄には「ファイルのコンテンツ」を入れると説明されています。では、先ほど「ファイルのコンテンツ」欄に入っていた「Id」はファイルのコンテンツなのでしょうか？

9.3 Case アップロードしたはずのファイルが開かない



図 9.19: ファイルのコンテンツ欄

動的な値から改めて「Id」を確認したところ、「ファイルの識別子」であり、「ファイルのコンテンツ」ではないことがわかりました。つまり、ファイルの作成アクションの「ファイル コンテンツ」の設定を間違っていたため、ドキュメントライブラリに正しくファイルが作成されていないということがわかりました。

9.3.9 ファイル コンテンツ は存在するのか？

ファイルの作成アクションよりも前のアクションで「ファイルのコンテンツ」が取得できていれば、ファイルの作成アクションの「ファイル コンテンツ」に「ファイルのコンテンツ」を入れることができます。では、今のフローで「ファイルのコンテンツ」を取得できているのでしょうか？ 動的な値を検索して、確認してみましょう。

第9章 フローで思い通りの結果を得るために大切なこと



図 9.20: 動的な値 1

9.3 Case アップロードしたはずのファイルが開かない

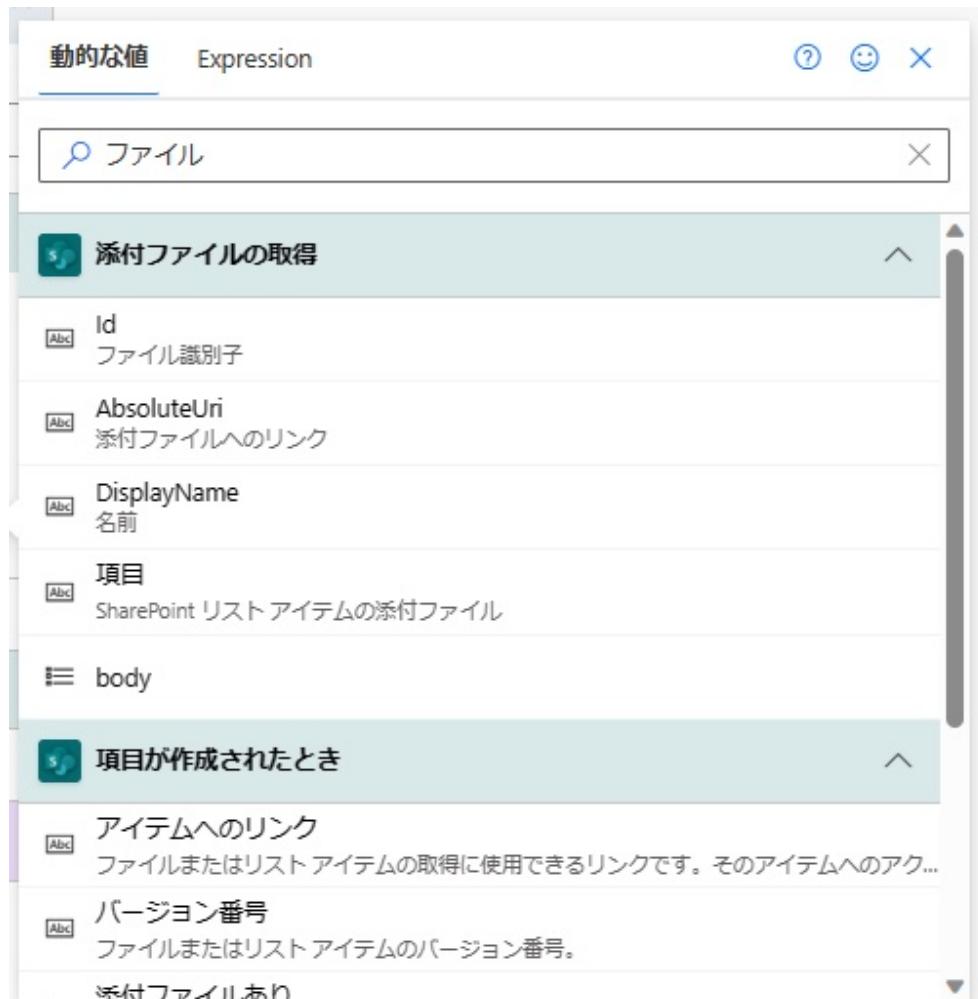


図 9.21: 動的な値 2

第9章 フローで思い通りの結果を得るために大切なこと

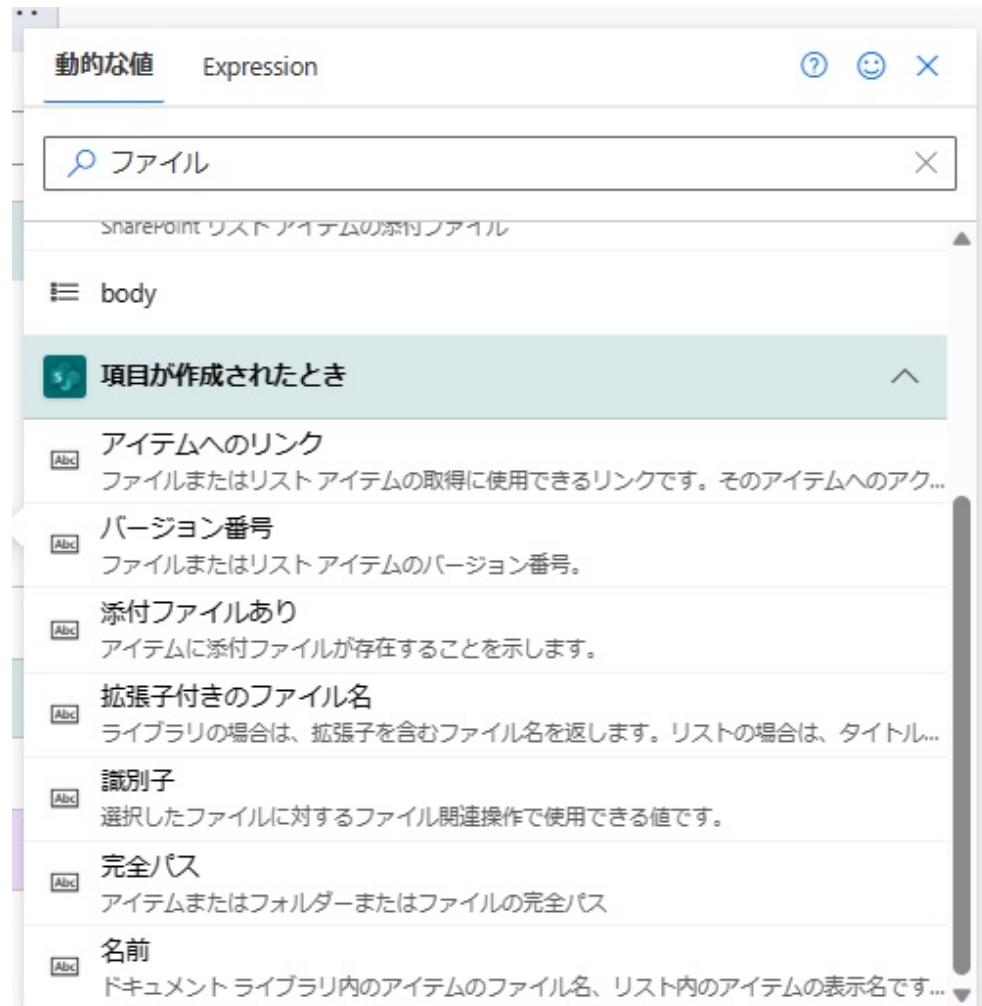


図 9.22: 動的な値 3

9.3 Case アップロードしたはずのファイルが開かない

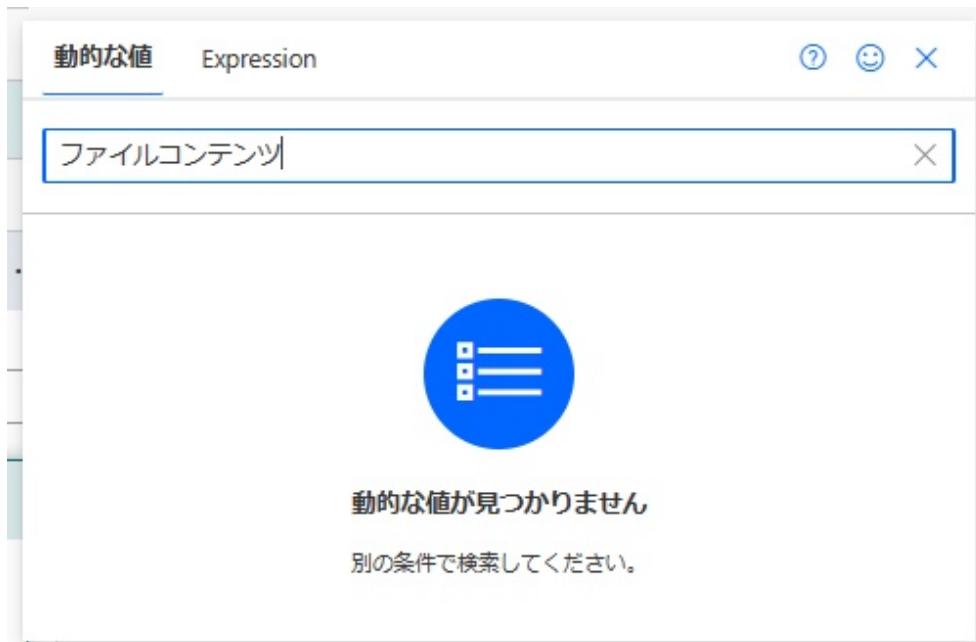


図 9.23: 動的な値 4

「ファイルの作成」アクションの「ファイル コンテンツ」欄にカーソルを当て、「コンテンツ」「ファイル」「ファイルコンテンツ」の 3 つで検索しましたが、「ファイルのコンテンツ」らしき動的な値は見つかりませんでした。つまり、新しくアクションを追加して、「ファイルのコンテンツ」を取得しなければならないということがわかりました。

9.3.10 どのアクションで「ファイルのコンテンツ」を取得するか

取得したいファイルのコンテンツは、レコードの添付ファイルのコンテンツです。ゆえにレコードの添付ファイルのコンテンツを取得するには、どうすればいいかを考える必要があります。わからないことがある時は、公式リファレンスに立ち返るのが 1 番です。今回も Microsoft Learn を読み解いていきたいと思います。

SharePoint コネクタのアクション一覧を見ていくと、「添付ファイルのコンテンツを取得」というアクションがあることがわかりました。

第9章 フローで思い通りの結果を得るために大切なこと

新しいドキュメントセットを作成する	新しいドキュメントセットリスト品目を作成します。
新しいフォルダーの作成	新しいフォルダまたはフォルダパスを作成します。
添付ファイルのコンテンツを取得	ファイル識別子を使用してファイルの内容を返します。 内容は別の場所にコピーすることも、添付ファイルとして使用することもできます。
添付ファイルの削除	指定された添付ファイルを削除します。
添付ファイルの追加	指定したリストアイテムに新しい添付ファイルを追加します。
添付ファイルを取得する	指定されたリストアイテムの添付ファイルのリストを返します。「添付ファイルの内容の取得」の手順を追加し、このアクションによって返される「ファイル識別子」プロパティを使用して、ファイルの内容を取得できます。

図 9.24: 添付ファイルのコンテンツを取得 (出典:Microsoft Learn)

これを使えば、添付ファイルのコンテンツが取得できそうです。Microsoft Learn のアクション名をクリックして、詳細ページに飛んでみましょう。

9.3.11 「添付ファイルのコンテンツを取得」アクションを理解する

初めて見るアクションは、Microsoft Learn で確認。ここまで読み進めた皆さん、きっと覚えられたと思います。

9.3 Case アップロードしたはずのファイルが開かない

添付ファイルのコンテンツを取得

操作 ID: GetAttachmentContent

ファイル識別子を使用してファイルの内容を返します。 内容は別の場所にコピーすることも、添付ファイルとして使用することもできます。

パラメーター

Name	キー	必須	型	説明
サイトのアドレス	dataset	True	string	例: https://contoso.sharepoint.com/sites/sitename 。
リスト名	table	True	string	SharePoint リスト名。
ID	itemId	True	integer	ファイルが添付されたリスト ID。
ファイル識別子	attachmentId	True	string	添付ファイルのファイル識別子。

戻り値

添付ファイルのコンテンツ。

添付ファイルのコンテンツ binary

図 9.25: 添付ファイルのコンテンツを取得 (出典:Microsoft Learn)

Microsoft Learn を読むと、ファイルが添付されたレコードの ID と 添付ファイルのファイル識別子がわかれば、添付ファイルのコンテンツを取得できるということが理解できます。

9.3.12 フローをコピーし、「添付ファイルのコンテンツを取得」アクションを追加する

いよいよ「添付ファイルのコンテンツを取得」アクションを追加して、「ファイル コンテンツ」を取得したいところですが、まずはフローをコピーしておきましょう。フローをコピーすることで現在のフローをそのまま残すことが可能になり、差分比較が行いやすくなります。フローは「名前を付けて保存」でコピーできます。

第9章 フローで思い通りの結果を得るために大切なこと

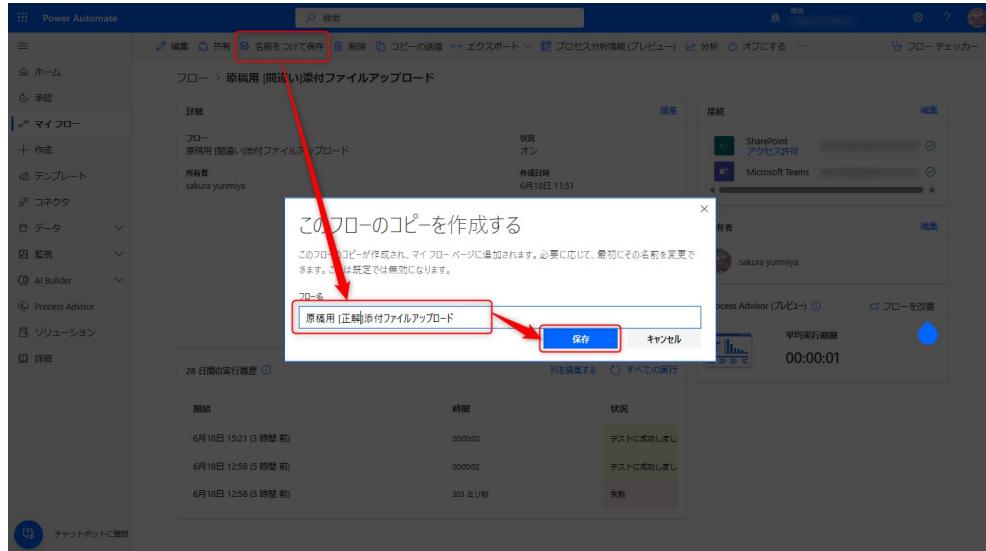


図 9.26: フローのコピー 1

コピーしたフローはデフォルトでオフになっていますので、オンにしてから編集しましょう。



図 9.27: フローのコピー 2

「添付ファイルのコンテンツの取得」アクションは、添付ファイルを1つずつ取得しな

9.3 Case アップロードしたはずのファイルが開かない

ければならないので、Apply to each アクションの先頭に入れます。

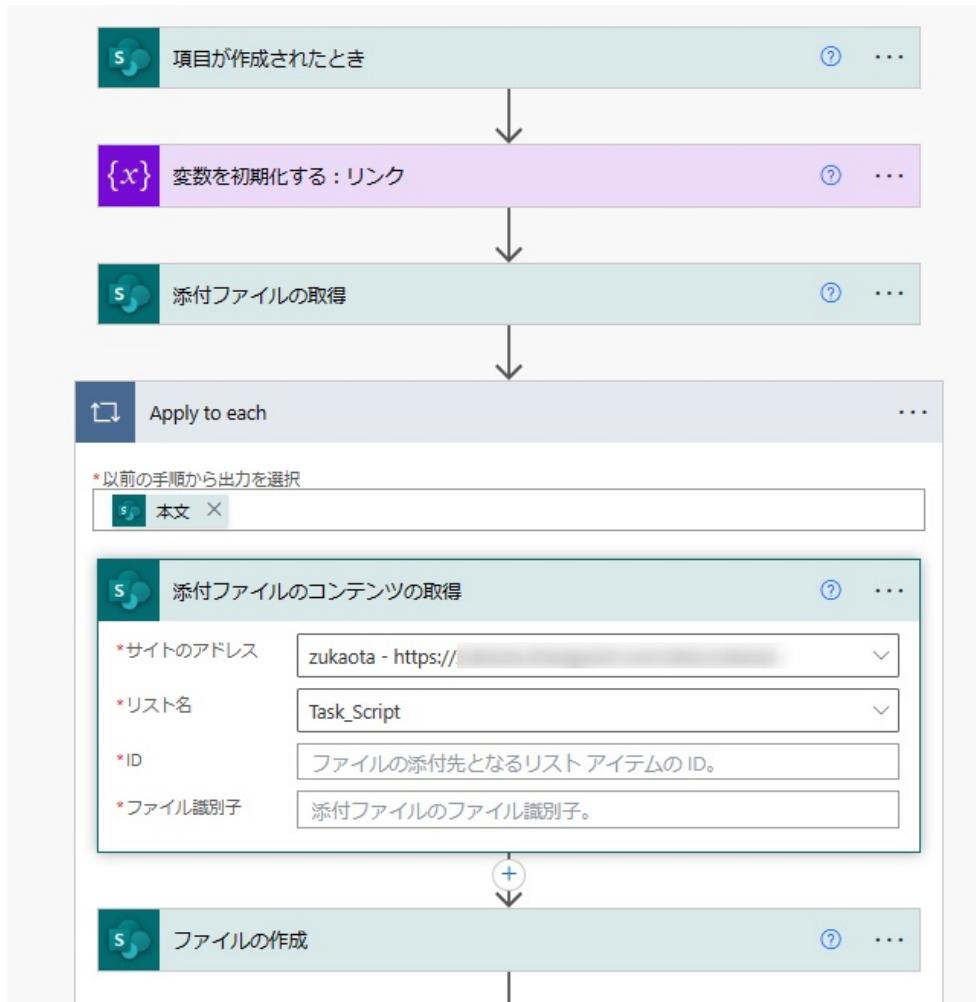


図 9.28: 添付ファイルのコンテンツの取得

9.3.13 ID と ファイル識別子が取得できるか確認する

では、先ほどのファイルのコンテンツと同様に、ID とファイル識別子が現在のフローで取得できているのかを確認しましょう。動的な値で「id」を検索した結果は画像の通りです。

第9章 フローで思い通りの結果を得るために大切なこと



図 9.29: 動的な値

項目名の下の説明で、「項目が作成されたとき」の「ID」が「ID」に、「添付ファイルの取得」の「Id」が「ファイル識別子」になりそうだということがわかりました。実際にアクションに動的な値を入れ込んでみると、こんな感じです。

9.3 Case アップロードしたはずのファイルが開かない

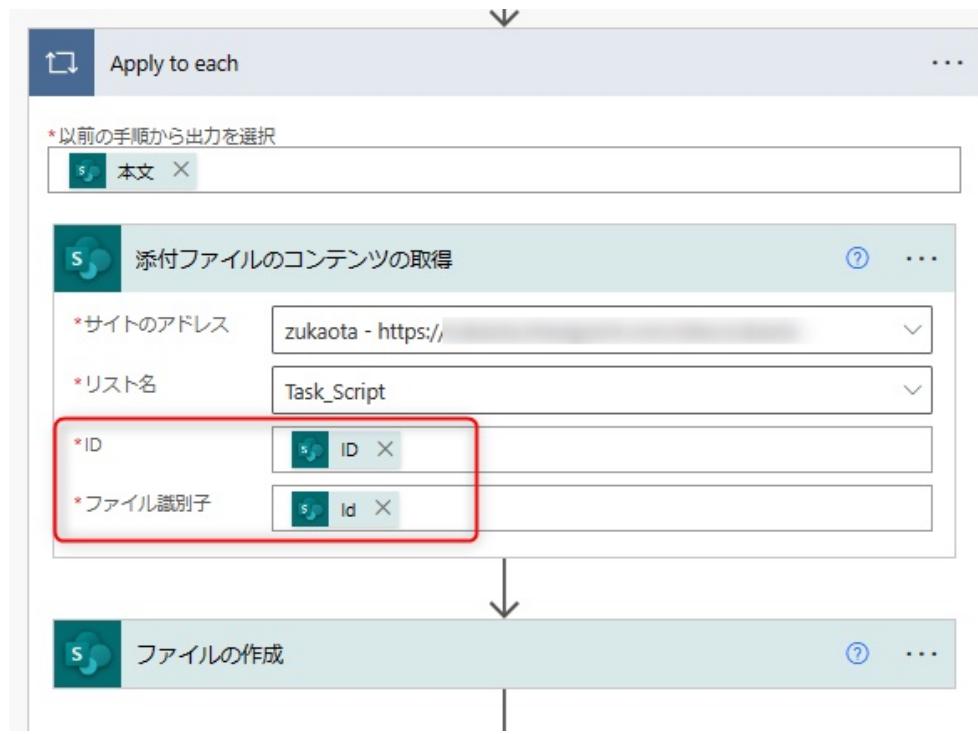


図 9.30: 添付ファイルのコンテンツの取得

9.3.14 「ファイルの作成」アクションの設定し直す

「添付ファイルのコンテンツの取得」アクションの設定が完了しました。これで「ファイルのコンテンツ」が取得できるようになるはずです。では改めて、「ファイルの作成」アクションの「ファイル コンテンツ」にカーソルをあてて、動的な値を検索してみましょう。

第9章 フローで思い通りの結果を得るために大切なこと



図 9.31: 動的な値

無事に「添付ファイルのコンテンツの取得」の「添付ファイルのコンテンツ」が出てきました。こちらを「ファイル コンテンツ」に入れていきましょう。

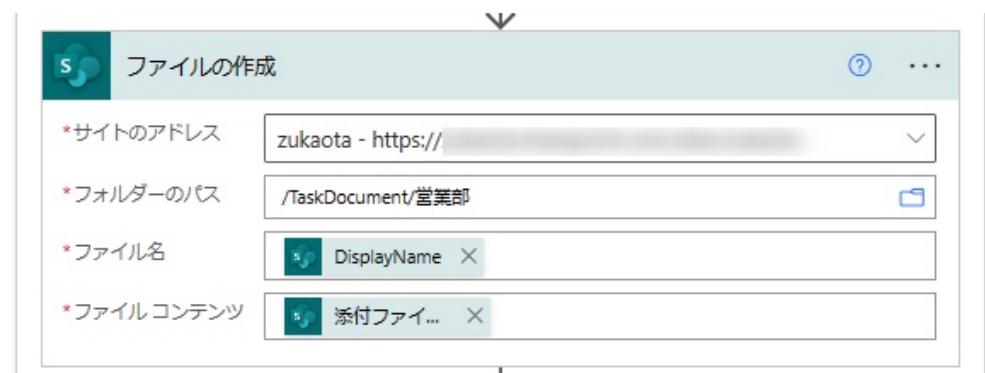


図 9.32: ファイルの作成

これで「ファイルが破損している可能性」を潰したことになります。テストを行っていきましょう。

9.3 Case アップロードしたはずのファイルが開かない

9.3.15 フローをテストし直す

テスト機能を使用して、再びフローの動作テストを行います。最初のテストと同一の環境で行うために、1回目のテストでドキュメントライブラリにアップロードしたファイルは、すべて削除しておきましょう。

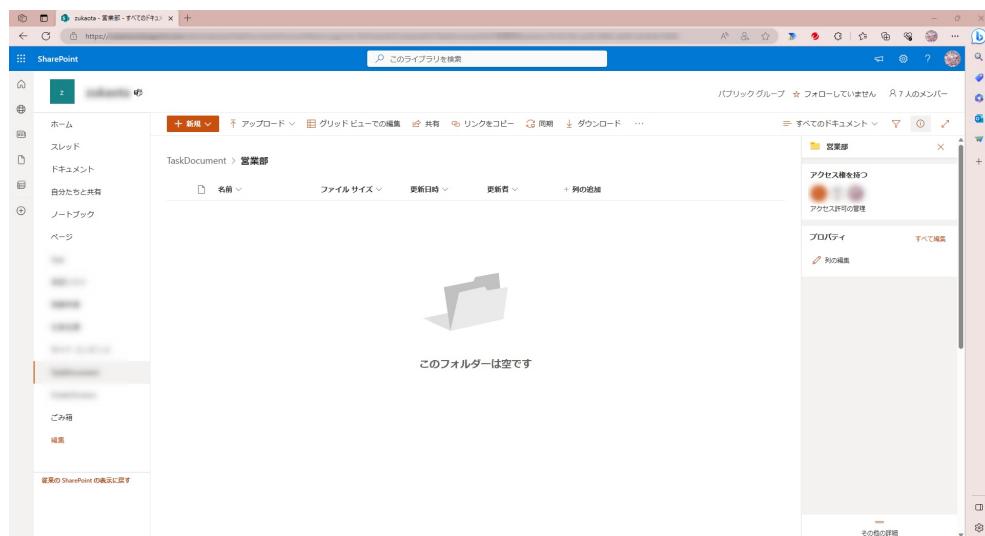


図 9.33: ドキュメントライブラリ

第9章 フローで思い通りの結果を得るために大切なこと

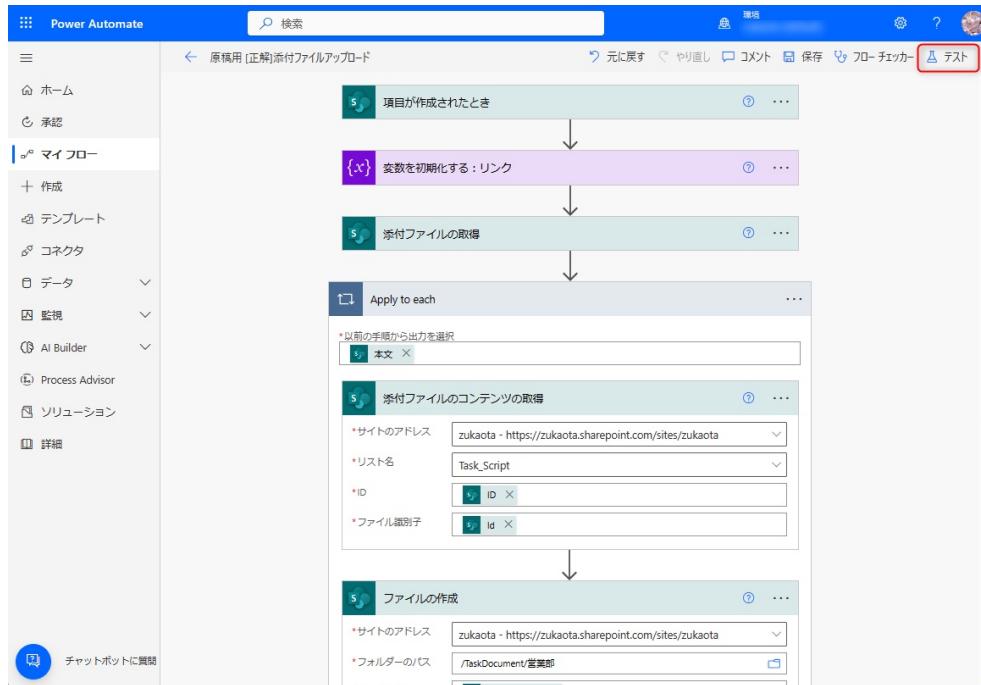


図 9.34: テストの実行

フローの成功が確認できました。

9.3 Case アップロードしたはずのファイルが開かない



図 9.35: フローの実行

前回同様、想定する「正常な動作」の確認をしていきます。まずは、Teams にメッセージが投稿されているかを確認します。

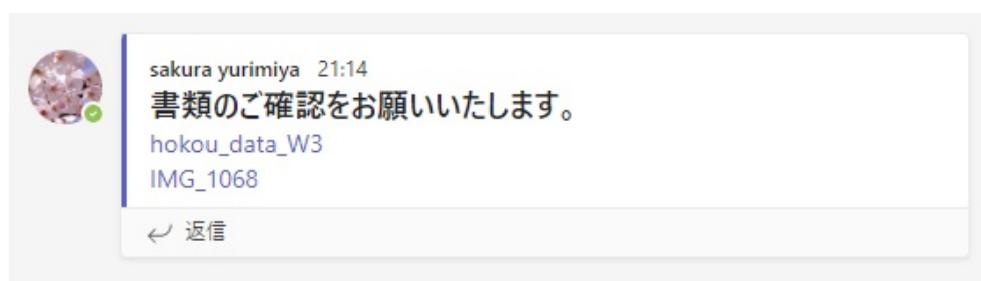


図 9.36: メッセージの投稿

きちんと投稿されていました。次にメッセージのリンクが開くかどうかを確認します。1つ目のリンク「hokou_data_W3」をクリックします。

第9章 フローで思い通りの結果を得るために大切なこと

The screenshot shows an Excel spreadsheet window with the title bar 'hokou_data_W3 - 保存済み'. The ribbon menu is visible with tabs like 'ホーム', '挿入', '描画', etc. The main area displays a table with columns labeled '家族構成', '売上月', and '売上'. The data consists of 15 rows of information, mostly for 'ファミリー' and '単身' households across April and May, with values ranging from ¥5,000 to ¥20,000. Row 20 is highlighted in green, indicating it is selected. The bottom of the screen shows the status bar with 'Microsoft にフィードバックを送信' and zoom controls.

家族構成	売上月	売上
2 ファミリー	4月	¥15,000
3 ファミリー	4月	¥20,000
4 単身	4月	¥12,000
5 ファミリー	4月	¥18,000
6 単身	4月	¥9,000
7 単身	4月	¥5,000
8 単身	4月	¥10,000
9 ファミリー	4月	¥17,000
10 単身	4月	¥15,000
11 ファミリー	5月	¥10,000
12 単身	5月	¥15,000
13 ファミリー	5月	¥13,000
14 ファミリー	5月	¥12,000
15 単身	5月	¥16,000

図 9.37: Excel ファイル「hokou_data_W3」の表示

Excel ファイルが問題なく開きました。2つ目のリンク「MG_1068」もクリックしてみます。

9.3 Case アップロードしたはずのファイルが開かない

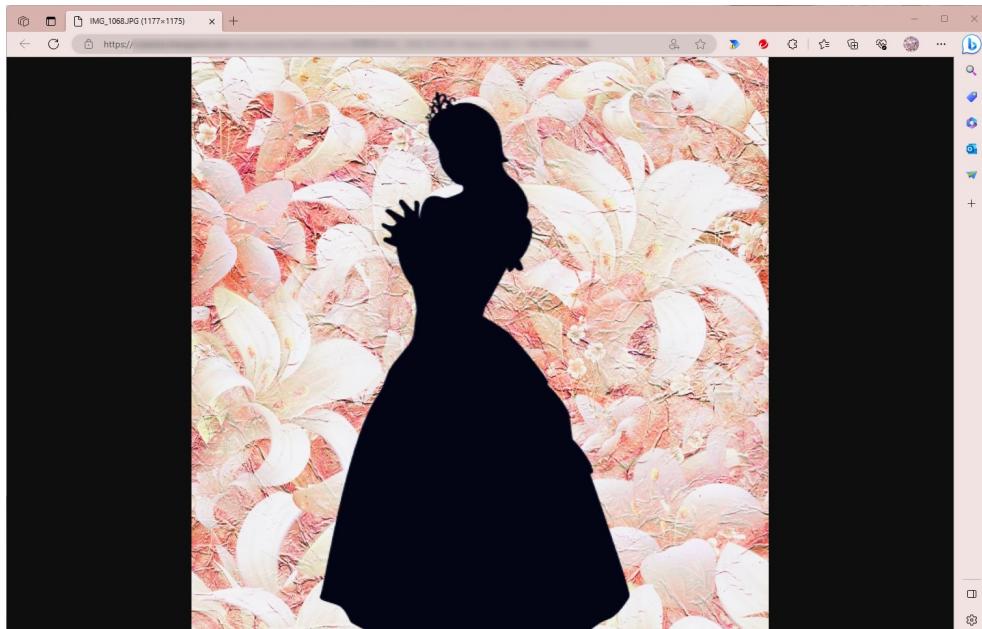


図 9.38: 画像ファイルの表示

こちらも問題なく開きました。ドキュメントライブラリ上のファイルサイズも確認しておきましょう。

第9章 フローで思い通りの結果を得るために大切なこと

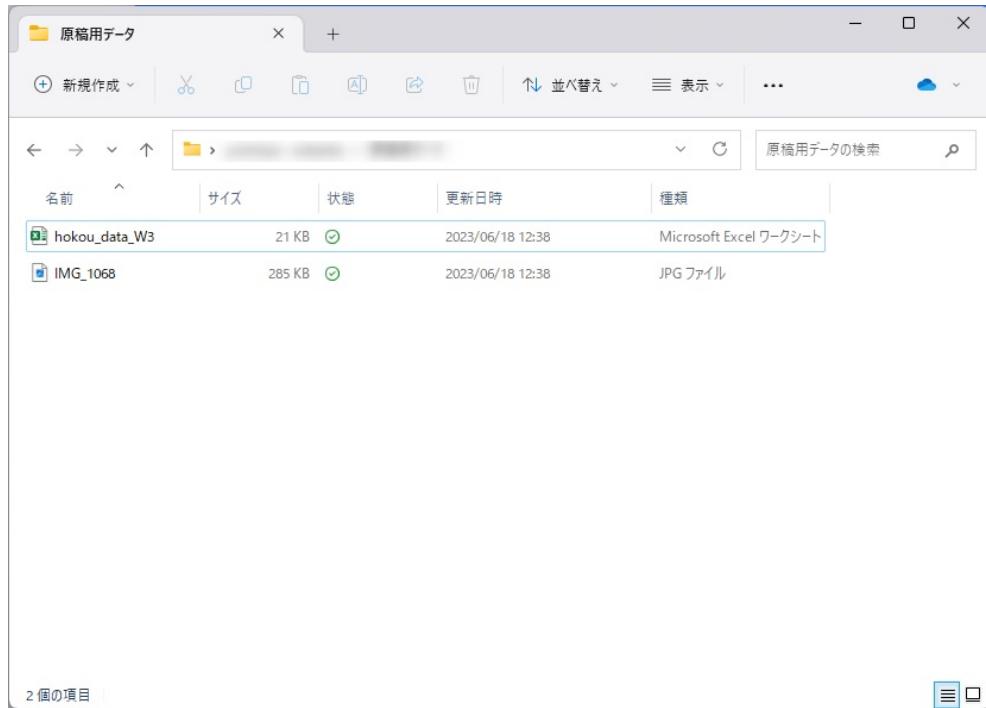


図 9.39: 元データ

TaskDocument > 営業部

名前	ファイルサイズ
hokou_data_W3.xlsx	20.8 KB
IMG_1068.JPG	285 KB

図 9.40: ドキュメントライブラリ上のデータ

元データとほぼ同じ大きさで、ファイルが作成されていることがわかりました。これでデバッグ完了です。作成者の想定通りに正しく動くフローができました。

9.4 バグのない正しいフローを作るために

9.4 バグのない正しいフローを作るために

バグのない正しいフローを作成するためには、テストが非常に重要です。アクション数が少ない小さなフローなら、頭の中でテスト項目を思い描いてテストすることも可能ですが、アクション数が多いフローや Power Apps など他のツールと連携するフローは頭の中でテスト項目を思い描いても、どうしても抜け漏れが出ることも多いです。そのため、バグを確実に潰した正しいフローを納品するにはテスト項目を洗い出し、1つずつ確認していくことが必要です。最後の効率よく抜け漏れのないテストを行うためのコツを書いていきます。

9.4.1 テスト項目ってどうやって洗い出すの？

Power Automate の場合は、想定通りの「正常な動作」が何かを考え、それを書き出して確認していくのが良いです。今回のフローの場合は、以下の通りです。S * SharePoint リストにレコードが作成された時にフローが実行される。* Teams にリンク付きメッセージが投稿される。* メッセージの中にあるリンクを開くことができる。

9.4.2 テスト方法と想定結果、実際の結果を一覧表にして確認する

テストを漏れなくきっちり行うためには一覧表にするとわかりやすいです。次の表は、最初のテスト結果をまとめてみました。

テスト方法	想定結果	実際の結果
フローの実行 SharePoint リストにレコードを新規作成する	フローが実行される	想定通り
	Teams にメッセージが投稿される	想定通り
メッセージのリンク確認 メッセージ内のリンクをクリックし、問題なくリンクが開くか確認する	リンクが開き、ファイルが表示される	リンクが開かない

図 9.41: テストの一覧表

テスト方法の太字部分はテスト方法のタイトル、下の文章はテストの方法を説明しています。想定結果がいくつかある場合は、テスト方法のセルを結合させて、ひとまとめに

第9章 フローで思い通りの結果を得るために大切なこと

見えるようにしています。この方法の良いところは、4つあります。

- Excelなどの表が作成できるソフトを使うと、不足分の付け足しが簡単。
- 方法と想定結果/実際の結果が1行にまとまって見やすい。
- 確認事項が記録されるので、抜け漏れがなくなる。
- チームでの確認が行いやすい。

あくまでも一例ですが、テストで迷った時はぜひやってみてください。きっと以前よりも安心感のあるフローが作成できると思います。

9.5 終わりに

今回、Power Automateでのバグの潰し方とテスト方法を紹介いたしました。この2つを本書で紹介しようと思ったのは、実は Microsoft Build がきっかけです。Microsoft Build では、Windows や Power Platform をはじめとする数多くの製品への AI 搭載が発表されました。

Power Automate でも米国プレビュー環境のみですが、Copilot が実装されており、フローの作成は自然言語のみで可能となりつつあります。しかし Copilot はあくまでも「副」操縦士です。作成者のあなたの思考ややりたいことを完璧にトレースできるわけではありませんし、時には間違った解釈をしてしまうこともあります。簡単に動くものが作成できる時代だからこそ、フローが「想定通り」に「正常な動作」をするかどうかを確認するテスト作業は今後ますます重要になってきます。

皆さんにはぜひ入念なチェックに時間を使っていただき、楽しく AI と共に存する世界の担い手となって頂きたいと思います。ここまでお読みいただき、ありがとうございました。

筆者紹介

北崎 恵凡 (きたさき あやちか)

趣味でモノづくり活動やコミュニティ「野良ハック」や技術書の執筆や月刊 I/O(工学社)、シェルスクリプトマガジン(USP 出版)などへの寄稿を行う。共著書に「Jetson Nano 超入門」(ソーテック社)、「現場で使える！ Google Apps Script レシピ集」(インプレス R&D)、「図解と実践で現場で使える Grafana」(インプレス R&D)、「はじめての Node-RED MCU Edition」(工学社)がある。コミュニティではオンライン・オフライン・ハイブリッドイベントの企画・運営・配信・アーカイブ編集/管理支援を担当。

山田 雄一 (やまだ ゆういち)

SIer での受諾開発 SE、社内 SE(内製型) を経て、現在はサポートエンジニア。技術同人誌を中心とした技術書の執筆や、たま～に月刊 I/O(工学社) のライターだったりもする。(主に近畿圏近郊のイベントレポートを担当) 共著書に「エンジニアのための見積もり実践入門」、「エンジニアのためのオンライン生活ガイドブック」、「エンジニアのための目標設定の技術」、「エンジニアのための「カジュアル面談のトリセツ」、「積み基板を作らないための電子工作入門」などがある。(全て ditflame 名義(株)インプレス刊)

鎌田 誠 (かまだ まこと)

地元工場の情シスを 20 年経験した後、現在はその経験を活かして、名古屋を拠点にネットワーク系の技術営業に従事。お客様の課題解決等を行う。最近はコミュニティ活動に積極的に取り組んでおり、RPA Community ではライトニングトーク支部の主催。 Babylon.js 勉強会では「Babylon.js レシピ集 vol.1」(インプレス R&D) 及び「Babylon.js レシピ集 vol.2」の執筆も一部担当。IT 知識は広く浅くをモットーに日々精進。

付録 筆者紹介

瀧川 大樹（たきがわ だいき）

学生時代は社会情報学を専攻、推薦アルゴリズムについて研究。強調フィルタリングとコンテンツベースフィルタリングを組み合わせた独自のアルゴリズムを考案、LAMPで実装。また、研究の傍らデータセンターのアルバイトでサーバーの保守運用業務を経験。2014年より通信会社にてサーバー/アプリケーション保守運用業務に従事。保守運用業務に注力する傍ら、情報処理安全確保支援士の資格を取得し、システムのセキュリティー管理業務にも従事。

青木 敬樹（あおき ひろき）

学生時代は機械工学を専攻。三次元画像の特微量抽出に関する研究。2021年に某通信会社に入社。入社後はサーバー・インフラの保守運用業務を担当。社内の「プロアクティブな運用」の推進チームの一員として活動し、運用可能な自動化を実現するために日々勉強中。

大野 泰歩（おおの やすほ）

学生時代は機械学習の中でも強化学習という分野について研究。RoboCup 2D Soccerに触れておりました。2020年に通信会社に入社し、サーバ・インフラの運用保守業務に従事する傍ら、運用保守業務を自動化するためのシステム開発にも関わっておりました。

松岡 光隆（まつおか みつたか）

XXX

新山 実奈子（にいやま みなこ）

XXX

百合宮 桜（ゆりみや さくら）

XXX

小崎 肇 (こさき はじめ)

フリーランス。COBOL から始まった 30 年以上のエンジニア経験を経て、Excel-VBA のノウハウと共に UiPath を使った業務効率化プロジェクトに参画。RPACommunity での登壇をキッカケに、運営側としても活動。定年退職後も UiPath 開発者として日々精進。RPA 界隈を賑やかしている中では、最高齢ではないかと玄いでいます。座右の銘は「人間万事塞翁が馬」。

澁谷 匠 (しぶや たくみ)

株式会社 ASAHI Accounting Robot 研究所 テクニカルエバンジェリスト。一般社団法人中小企業個人情報セキュリティー推進協会認定 DX アドバイザースペシャリスト。プログラミング知識ゼロの状態から RPA/AI-OCR の技術・知識及び概念、全社展開の導入スキーム等を独学で勉強。モットーは "Make everyone's work easier. (全ての人の仕事を楽にする)"。

慈英 (じえい)

XXX

現場で使える!自動化入門

2023年7月31日 初版第1刷 発行

著 者 北崎 恵凡、山田 雄一、鎌田 誠、瀧川 大樹、青木 敬樹、大野 泰歩、松岡 光
隆、新山 実奈子、百合宮 桜、小崎 肇、濱谷 匠、慈英
