インプレスR&D［NextPublishing］

技術の泉 SERIES
E-Book / Print Book

# 個人開発をはじめよう!

クリエーター25人の実践エピソード

ゆずたそ 編


手軽に始めるコツやノウハウ、
便利ツールや開発テクニックも紹介!
アイディア出しからリリースまでの流れがわかる!

impress R&D
An impress Group Company

GIJUTSU no IZUMI SERIES
泉
POWERD by NEXTPUBLISHING
技術の泉 SERIES

インプレスR&D［NextPublishing］
技術の泉 SERIES
E-Book / Print Book

# 個人開発をはじめよう!

クリエーター25人の実践エピソード

ゆずたそ 編


手軽に始めるコツやノウハウ、
便利ツールや開発テクニックも紹介!
アイディア出しからリリースまでの流れがわかる!

impress R&D
An impress
技術の泉 SERIES

## 電子書籍閲覧に関するご注意

本書では、プログラムリストに専用の等幅フォントを使用しています。ビューアによって以下の作業が必要になります。

・Kindle Paperwhiteの場合：フォント設定画面で「出版者のフォント」を選択
・kobo Androidアプリの場合：フォント画面で「オリジナル」を選択

# 目次


電子書籍閲覧に関するご注意

まえがき

- はじめに
- お問い合わせ先
- 免責事項

第1章　未経験から3ヶ月でつくる！個人開発マニュアル

- 1.1　1か月目 サービス設計と基礎知識の学習
- 1.2　2か月目 エラーで行き詰まってもくもく会に参加しまくる
- 1.3　3か月目 70%の完成度でリリースする
- 1.4　まとめ

第2章　とがったテーマの婚活サイトを開発して得た経験

- 2.1　作った動機
- 2.2　開発中に苦労したこと①：技術レベルが足りない！
- 2.3　開発中に苦労したこと②：デザインができない！
- 2.4　開発中に苦労したこと③：リリースに踏み切れない！
- 2.5　開発中の良かったこと
- 2.6　リリース後の大変だったこと
- 2.7　リリース後の嬉しかったこと
- 2.8　最後に

第3章　アイデアからリリースまで、Webサービス開発を徹底詳説

- 3.1　サービス紹介 - ありそうでなかったWEBサイト
- 3.2　企画 - ユーザーと向き合って考える
- 3.3　開発 - 小さく始めるアグリゲーションサービス
- 3.4　リリース - 価値を届ける
- 3.5　まとめ -「私の個人開発」奮闘記

第4章　「通知止まらんwww」体験アプリを作ったらバズって通知止まらんwww

- 4.1　はじめに
- 4.2　作ったもの
- 4.3　苦労した点
- 4.4　まさかの大反響
- 4.5　おわりに

第5章　積読の解消を促すWebサービスをリリースした話

- 5.1　サービス内容
- 5.2　きっかけとコンセプト
- 5.3　使い方
- 5.4　ランキング機能
- 5.5　Twitterでシェアして積読金額を晒す機能
- 5.6　リリースした後のプロモーション

5.7　登録サイトに投稿・依頼
5.8　いろんなところで記事を執筆
5.9　進捗・使い方・Tipsに関する投稿
5.10　おわりに

第6章　累積120万DL！なぜかインドで大ヒット！クソアプリを量産しよう！
6.1　真実はいつも1つ！「蝶ネクタイ型変声機」
6.2　ほろびこそわがよろこび！「RPG風画像ジェネレーター」
6.3　おらワクワクすっぞ！「瞬間移動」
6.4　野球好きなあなたに！「バッティングセンター ～無限素振り～」
6.5　サムライが好きなあなたに！「侍刀 ～The Samurai Sword～」
6.6　鳴らして遊ぼう！「ジングルベル」
6.7　正真正銘のクソアプリ！「つんつくつん」
6.8　埼玉の食卓から、世界のShokutakuへ
6.9　あくまでクソアプリ

第7章　ダウンロード数と収入報告！スマホアプリの個人開発は儲かるのか？
7.1　スマホアプリの個人開発は儲かるのか？
7.2　アプリ開発でのお金の稼ぎ方
7.3　アプリのダウンロード数と収入報告
7.4　ダウンロード数の考察
7.5　有料アプリの必要性について
7.6　アプリ開発は転職に有利
7.7　アプリ開発の壁
7.8　アプリ開発で稼ぐために
7.9　まとめ
7.10　最後に

第8章　格安スクレイピングを支える技術
8.1　はじめに
8.2　このパートで学べること
8.3　スクレイピングで気をつけるべきこと
8.4　スクレイピングを行うプログラムの基本構造
8.5　スクレイピングが捗るライブラリーの紹介
8.6　おまけ：オススメのインフラ
8.7　あとがき

第9章　個人開発だけで食べられるようになるまでにやったこと
9.1　自己紹介
9.2　付箋アプリを作った理由
9.3　インストールしてもらうためにやったこと
9.4　使い続けてもらうためにやったこと
9.5　オシャレなUIにするためにやったこと
9.6　収益を得るためにやったこと
9.7　安全で効率的な開発のためにやったこと
9.8　プロ個人開発者の働き方

9.9　プロ個人開発者の考え方

第10章　ストアからアプリが削除！消された個人開発アプリとその理由を公開！

10.1　アプリの開発歴や実績はどのくらいあるか
10.2　これまでどんなアプリを作ってきたのか
10.3　どんなアプリを消されてしまったか
10.4　事例1：[警告]コンテンツレーティングが合っていない
10.5　事例2：[警告]プライバシーポリシーがない
10.6　事例3：[削除]イラストの露出度が高い（Part1）
10.7　事例4：[削除]イラストの露出度が高い（Part2）
10.8　事例5：[削除]広告の表示方法が望ましくない
10.9　Google先生に怒られないようにするためには

第11章　自分の漫画ライフを充実させる「新刊通知アプリ」を作ってみた

11.1　どのようなアプリを作ったのか
11.2　なぜ作ったのか
11.3　どのように作ったのか
11.4　どのような苦労があったか
11.5　どのような学びがあったのか

第12章　カメラアプリを開発して憧れのレビューサイトに掲載された話

12.1　遅延再生カメラ
12.2　レビューサイトでの紹介
12.3　ダウンロード数は……
12.4　まとめ

第13章　会社が潰れて1ヶ月でリリース＆起業！エンジニア社長の奮闘記

13.1　はじめに: ある日のPitPaへの音声投稿
13.2　1st Mission - 1ヶ月でアプリをリリースせよ
13.3　2nd Mission - 10万再生を目指せ
13.4　3rd Mission - グロースハックせよ
13.5　4th Mission - イノベーションを起こせ
13.6　おわりに: ある日のPitPaへの音声投稿

第14章　個人開発のモチベーションが続かない、作り終わらない。その原因と対策

14.1　クリエイターの創作活動を応援するサイト「ためしがき」
14.2　クリエイターの創作活動を挫く「どうしてこうなった」の数々
14.3　原因1: 構想がでかすぎて作りきれない
14.4　原因2: 技術的にしんどい
14.5　原因3: 作ってる間に似たようなサービスが先に出てきた
14.6　原因4: 他のメンバーが動かないor自分の負担が大きい
14.7　原因5: 作ったけど誰にも使われなくて悲しい
14.8　まとめ

第15章　学生エンジニアが語る「技術学習 x 個人開発」のススメ

15.1　自己紹介

15.2　React と恋に落ちた理由
15.3　Study Typer とは
15.4　技術的なお話 ～どうやって作ったか～
15.5　システム構成
15.6　英単語データ
15.7　Reactの実装
15.8　最後に

第16章　「このままでいいの？」駆動のサービス開発

16.1　地域の魅力をクイズにする「Korette」
16.2　諦めかける日々との戦い
16.3　「このままでいいの？」を持つということ
16.4　アイデアはどうやったら思いつくのか
16.5　サービスを開発したいと思っている人へ
16.6　Koretteのこれから
16.7　あなたの「このままでいいの？」は何ですか？

第17章　使われなくてさみしい問題を解決する（しない）方法論

17.1　Webサービスの死因
17.2　個人開発サービスの一生
17.3　失敗例：book loves music（ブック・ラブズ・ミュージック）
17.4　「バズか死か」の二元論を避ける
17.5　【1】CGMではなくツール型にする
17.6　具体例：THE TOURNAMENT（ザ・トーナメント）
17.7　【2】できるだけ何も作らない
17.8　具体例：ブンゴウメール
17.9　具体例：THE TIMELINE
17.10　【3】絶対に赤字にはしない
17.11　具体例：スマホde百人一首
17.12　具体例：ゾラサーチ
17.13　死なないだけだと、ただのゾンビ
17.14　最後は確率論、いっぱい作ろう

第18章　サービス開発のハードルが高いあなたに！便利プログラム開発のススメ

18.1　どんなプログラムを作ったの？
18.2　なんで作ったの？
18.3　システム構成
18.4　開発中のトラブルと反省
18.5　便利機能を開発してみてよかったこと
18.6　振り返ってみて

第19章　アクセスログから見えた時代の移り変わり

19.1　サービス紹介
19.2　モチベーション
19.3　AWSの構成
19.4　コミュニケーション促進

19.5　アクセスとツイート
19.6　ちょっと特別な経験

第20章　Pythonと野球オープンデータで地図アプリを作る
20.1　対象読者
20.2　きっかけは「もくもく会」参加
20.3　野球オープンデータ「Retrosheet」
20.4　野球（MLB）データベースを作る
20.5　インフラ構築を自動化
20.6　試しにPythonでデータを使ってみる
20.7　データを使ってやりたいこと
20.8　球場データを探す
20.9　Beautifulsoupでスクレイピング
20.10　geopyでらくらくGeocoding
20.11　bottle + Google Map APIでサクッっと地図アプリを作る
20.12　地図アプリの完成
20.13　まとめ

第21章　困りごとを1日で解決する「どやっ」駆動のツール開発
21.1　どんなツールを作ってるか
21.2　個人開発ソフトウェアを一挙紹介
21.3　「えいや！」→「どやっ！」→「いいね！」の成功体験
21.4　気をつけているポイント

第22章　10年つづくWEBサービスの育て方 ～地道な宣伝から法人契約まで～
22.1　街の電源検索サイト「モバイラーズオアシス」
22.2　サービス開発のきっかけ
22.3　サービス公開も反応なし、地道なデータ入力
22.4　サイトで扱っているデータを法人に販売
22.5　サービスが手のひらから飛び立つとき

第23章　Build to Think！
23.1　15年間を振り返って
23.2　つくりたいからつくる
23.3　個人開発と創意工夫
23.4　Build to Think
23.5　オフラインで閲覧できるブログ
23.6　Build to Think のサイクルを広げる
23.7　みんなでThink！
23.8　おわりに

第24章　14年かかった！個人開発で月収100万達成した話
24.1　自己紹介
24.2　実際の月間PVと月収
24.3　バズった経験
24.4　ランニングコスト

24.5　月収100万円で見えてきた事
24.6　失敗を重ねて得た知見
24.7　アイデア編
24.8　システム編
24.9　保守はツライ
24.10　メンテナンス編
24.11　マーケティング編
24.12　トラブル編
24.13　マネタイズ編

第25章　それでも挑戦は止められない。個人開発の絶望と希望
25.1　純粋な個人開発は楽しい
25.2　目的に向かって進むための個人開発
25.3　Webサービス作りをスタート
25.4　転機
25.5　一番大きな学び
25.6　ひたすら突っ走り始めた

付録A　141人に聞いた！みんなの個人開発事情

あとがき
関係者へのお礼
商業出版に寄せて
同人版との違い
今を生きるクリエイターに
「+1」
個人開発をはじめよう！

# まえがき

## はじめに

本書を手にとっていただき、ありがとうございます。この本は、個人でWebサービスやアプリを開発する「個人開発者」の合同誌です。「個人開発の楽しさ」をお伝えします。

想定読者①：プログラミング初学者

本書は先輩開発者のエピソード集です[1]。モチベーションの向上にお役立てください。「こういうアイデアを実現できたら楽しそうだ」という未来を描いてください。

「一度は自分でサービスをリリースしてみたい」という憧れがあるからこそ、本書を手に取ってくださったのではないでしょうか。筆者も同じです。先人の背中に憧れて個人開発を始めました。筆者にとって本書は「過去の自分」へのエールなのです。

想定読者②：個人開発に挫折しそうな（あるいは挫折してしまった）方々

本書は現役開発者のエピソード集です。再挑戦のキッカケとしてお役立てください。「こういうアイデアを実現するのが楽しいんだよな」と過去を思い返してください。

「もう一度頑張ってみたい」「あの頃は楽しかったな」という思いがあるからこそ、本書を手に取ってくださったのではないでしょうか。筆者も同じです。1年後には挫折しているかもしれません。筆者にとって本書は「未来の自分」へのエールなのです。

想定読者③：今まさに打ち込んでいる個人開発者

本書はあなたと同じ個人開発者のエピソード集です。「こういうことあるよね」と共感できるはずです。「これはマネしてみたい」と発見できるはずです。

自分で実践しているからこそ、共感や発見を求めて、本書を手に取ってくださったのではないでしょうか。筆者も同じです。同じ気持ちを持つクリエイターたちが集まって本書ができたのです。筆者にとって本書は「現在の自分」へのエールなのです。

本書の構成：「7つのテーマ」と「25人のエピソード」

> 「個人開発をはじめよう！」

そう思えるような、クリエイター25人の実践エピソードを集めました。

**【テーマ①】何からやればいい？ 個人開発のはじめ方。**

・第01章：未経験から3ヶ月でつくる！個人開発マニュアル

・第02章：とがったテーマの婚活サイトを開発して得た経験

・第03章：アイデアからリリースまで、Webサービス開発を徹底詳説

**【テーマ②】皆に使ってもらえる？ 個人開発で、バズった話。**

・第04章：「通知止まらんwww」体験アプリを作ったらバズって通知止まらんwww

・第05章：積読の解消を促すWebサービスをリリースした話

・第06章：累積120万DL！なぜかインドで大ヒット！クソアプリを量産しよう！

**【テーマ③】ぶっちゃけどうなの？ 個人開発と、おさいふ事情。**

・第07章：ダウンロード数と収入報告！スマホアプリの個人開発は儲かるのか？

・第08章：格安スクレイピングを支える技術

・第09章：個人開発だけで食べられるようになるまでにやったこと

**【テーマ④】試行錯誤の繰り返し？ こんな苦労も、個人開発。**

・第10章：ストアからアプリが削除！消された個人開発アプリとその理由を公開！

・第11章：自分の漫画ライフを充実させる「新刊通知アプリ」を作ってみた

・第12章：カメラアプリを開発して憧れのレビューサイトに掲載された話

・第13章：会社が潰れて1ヶ月でリリース＆起業！エンジニア社長の奮闘記

**【テーマ⑤】くじけそうなときは？ こうして続ける、個人開発。**

・第14章：個人開発のモチベーションが続かない、作り終わらない。その原因と対策

・第15章：学生エンジニアが語る「技術学習 x 個人開発」のススメ

・第16章：「このままでいいの？」駆動のサービス開発

・第17章：使われなくてさみしい問題を解決する（しない）方法論

**【テーマ⑥】作りたいものがない？ もっと自由に、個人開発。**

・第18章：サービス開発のハードルが高いあなたに！便利プログラム開発のススメ

・第19章：アクセスログから見えた時代の移り変わり

・第20章：Pythonと野球オープンデータで地図アプリを作る

・第21章：困りごとを1日で解決する「どやっ」駆動のツール開発

**【テーマ⑦】その先に見えるのは？ 個人開発と10年間。**

・第22章：10年つづくWEBサービスの育て方 ～地道な宣伝から法人契約まで～

・第23章：Build to Think！

・第24章：14年かかった！個人開発で月収100万達成した話

・第25章：それでも挑戦は止められない。個人開発の絶望と希望

7つのテーマに分かれていますが、好きなページを好きなように読んでいただければと思います。

本書の特徴：「事例集」であること

本書の内容は十人十色です。どう企画するか。どうモチベーションを保つか。どうユーザーを獲得するか。どうテクノロジーを使うか。様々な意見が次々と出てきます。

メディアで脚光を浴びる著名人に「これが正解だ」と教えてもらう本ではありません。悩みや迷いや失敗も含めて、等身大のエピソードを、1人1人の言葉で綴った本です。

ある人にはある人の楽しさがあります。別の人には別の人の楽しさがあります。それが個人開発なのだと思います。

あなただけの「個人開発」と出会える本

「個人開発」という言葉の定義は1人1人違います。しかし少なくとも本書に出てくるエピソードの中に「誰かに強制されたもの」はひとつもありません。やりたいときに、やりたいことを、やりたいようにやる。自分なりの正解を見つけていくのです。

筆者たち1人1人は、各々が考える「楽しさ」を書き記しました。ぜひ自分に合うものを見つけてもらいたいと思っています。「自分はこの考え方やエピソードが好きだ」「自分もこういうクリエイターでありたい」と少しでも感じてもらえたら嬉しいです。読者であるあなた自身が、そうした新しい可能性と出会える1冊の本になっていれば幸いです。

（発行代表・企画者 @yuzutas0）

## お問い合わせ先

本書に関するお問い合わせ: @yuzutas0（ゆずたそ）

## 免責事項

1．本書の内容は、筆者各人の個人的見解であり、所属組織を代表するものではありません。もし誤りや考慮不足だと感じる点があれば、それは編集・発行代表である @yuzutas0 の力不足によるものです。誠に恐縮ですが、上記宛先までご連絡いただけると幸いです。
2．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。本文中では © 、 ® 、 ™ マークなどは表示していません。
3．本書は有志による情報提供を目的とします。本書の利用により発生したいかなる損害に対しても筆者一同はその責任を負いかねます。特にシステム開発・運用におきましては、他の情報源をご確認の上、ご自身の責任と判断にもとづいて実施してください。
4．本書に記載されているサンプルコードや画面イメージ、URL等は執筆時点のものです。執筆以降で変更されている可能性があります。あらかじめご了承ください。

1. もしあなたがプログラミング教室や初学者向けコミュニティの運営者であれば、ぜひ学習支援コンテンツとして本書をご活用ください。お問い合わせいただければ可能な範囲でサポートを提供します。

# 第1章　未経験から3ヶ月でつくる！個人開発マニュアル

　フリーのWebエンジニアのかしいと申します。2018年11月にお笑いライブの検索サイト「ワラリー！」をリリースしました。このサービスを作りたくてプログラミングスクールに入会したのですが、1ヶ月後に気づきました。「個人開発やること多っ！！」

　このハードルを下げたいと思い、初めて個人開発したときの手順やコツをまとめました。

## 1.1　1か月目 サービス設計と基礎知識の学習

　きっかけは、お笑いライブの検索サイト「Walive!」がとつぜん閉鎖したことでした。お笑いファンに愛されているサービスが無くなってしまうのはもったいない。そう思った私は、サイトを引き継ぐためにサービス設計とプログラミングの学習を始めました。

**アイデア出し**

１．自分が**困っていること**をベースに考える（自分が欲しいサービスを作りたい）

２．自分が**今できること**をベースに考える（技術的な学びを実践することが目的）

３．**他人が困っていること**をベースに考える（稼ぎたい、ユーザーの反応を見たい）

　まずは1番がおすすめです。2番や3番だと、途中で「本当にみんなサービスを使ってくれるだろうか」と不安になります。自分が使うサービスのほうがモチベーションを維持しやすいです。

**サービス内容を決める**

　サービス内容は「ひとことで言える」ことが大事です。機能が多すぎると、結局何のためのサービスか分からなくなります。開発の負担も増えます。まずはひとつの機能に絞りましょう。

×「お笑いライブを検索できて、芸人のプロフィールを見られて、チケットを融通できるサービス」

○「お笑いライブを横断検索できるサービス」

**サービス名を決めて、ドメインを取得する**

　ドメインを取るためにサービス名を考えます。ポイントは下記の5つです。

・1. サービス内容を端的に言う：サービス内容そのものをサービス名にすれば、わかりやすさはピカイチです（例：チケットトレード→チケトレ、小説を書く・読む→カクヨム）。

・2. 英単語を組み合わせる：サービスで重要となる要素を洗い出し、語感がしっくりくるものを選びます。お笑いライブ検索「ワラリー」という名前は、wara（わらい）+rally（集まり） という英単語の組み合わせから決めました。

・3. 長いサービス名にしない：サービス名が長いとユーザーは覚えられません。略称でつけると良いでしょう（例：careertrek → キャリトレ）。

・4. 日本語の方が親しみやすい：難しすぎる英語だと覚えられません。読めなかったりもします。カタカナでつけてしまいましょう。

・5. ユニークにする：名称が被るとエゴサーチで反響をチェックしづらくなります。

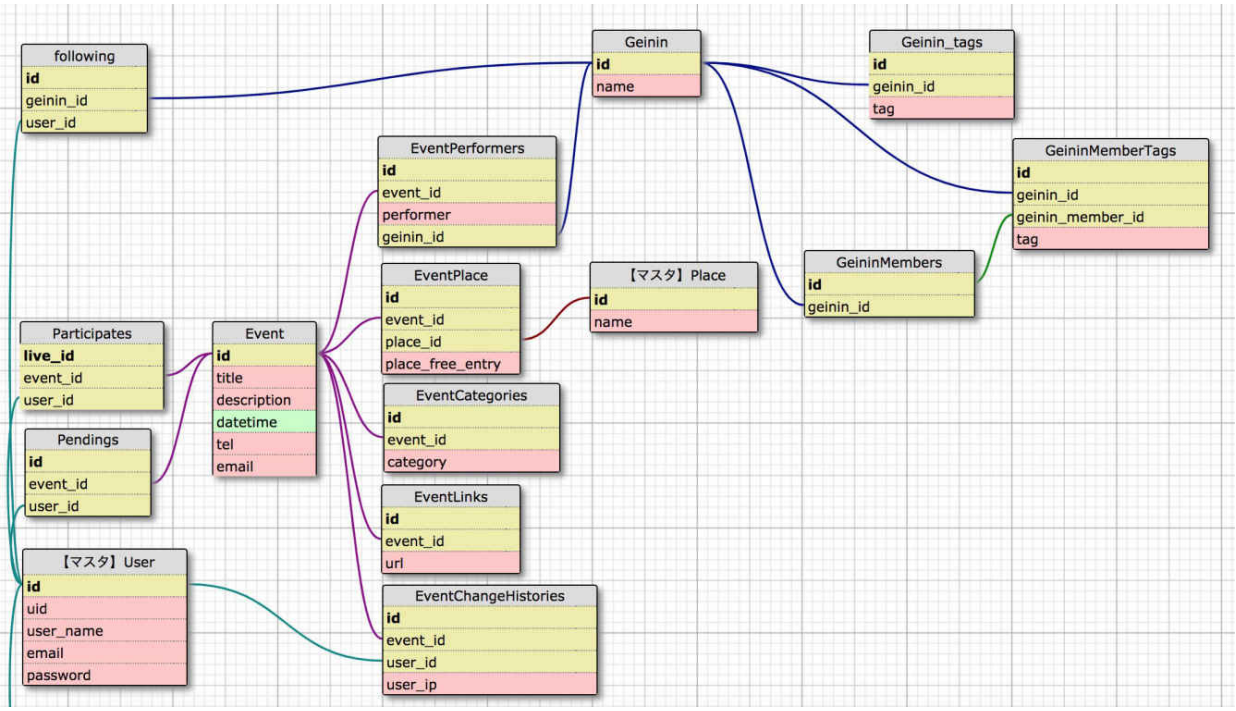
### サービスで使う技術を決める

　初心者なら、まずはRubyやPHP、HTML/CSSとjQueryを学びましょう。フレームワークは情報を探しやすいRuby on RailsかLaravelがおすすめです。

### モデル（テーブル）設計

　「SQL designer」というWebブラウザーアプリを使うと、ER図を簡単に作成できます。

図1.1: ER図




　テーブル同士の関係は親と子までにすると良いでしょう。初心者向けの教材だと親と孫、ひ孫のデータの扱い方まで解説している記事が少なく、つまづくポイントとなります。

　「実際に運用可能か」ということも考慮ポイントとなります。たとえばワラリー！では、お笑いライブのチケット予約先を複数記録するためにふたつのモデルを作りました。

　・Eventモデル（お笑いライブの情報を記録する親モデル）

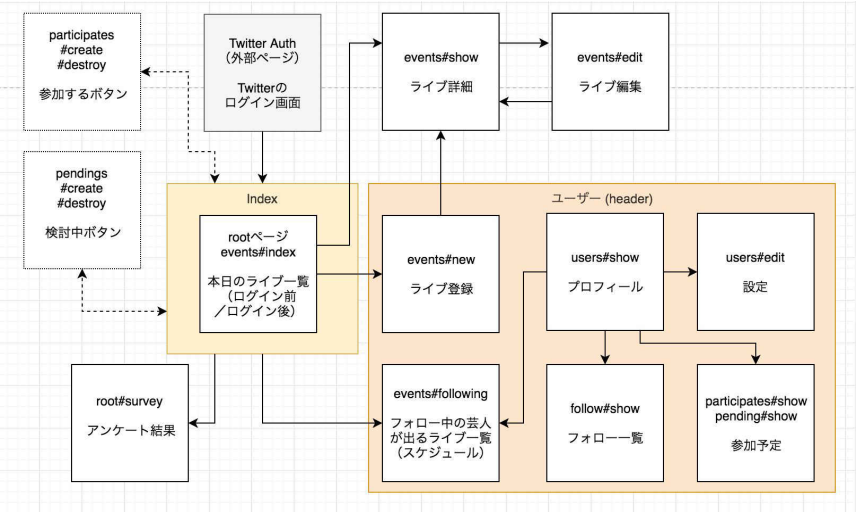
　・EventLinkモデル（お笑いライブのイベントの予約URLを記録する子モデル）

　しかし今考えると、最初はEventモデルに「URL」というカラムをひとつ作成すれば十分だったと思っています。手入力でわざわざ複数の予約サイトを登録するのは手間なのです。扱える情報を制限する方が、使いやすくなるという教訓を得ました。

### 画面遷移の設計

ユーザーがどのような順番でページにアクセスするかを考えます。私は「Draw.io」を使って画面遷移図を作成しましたが、紙でも十分だと思います。

図1.2: 画面遷移図




**サービスのデザインを決める**

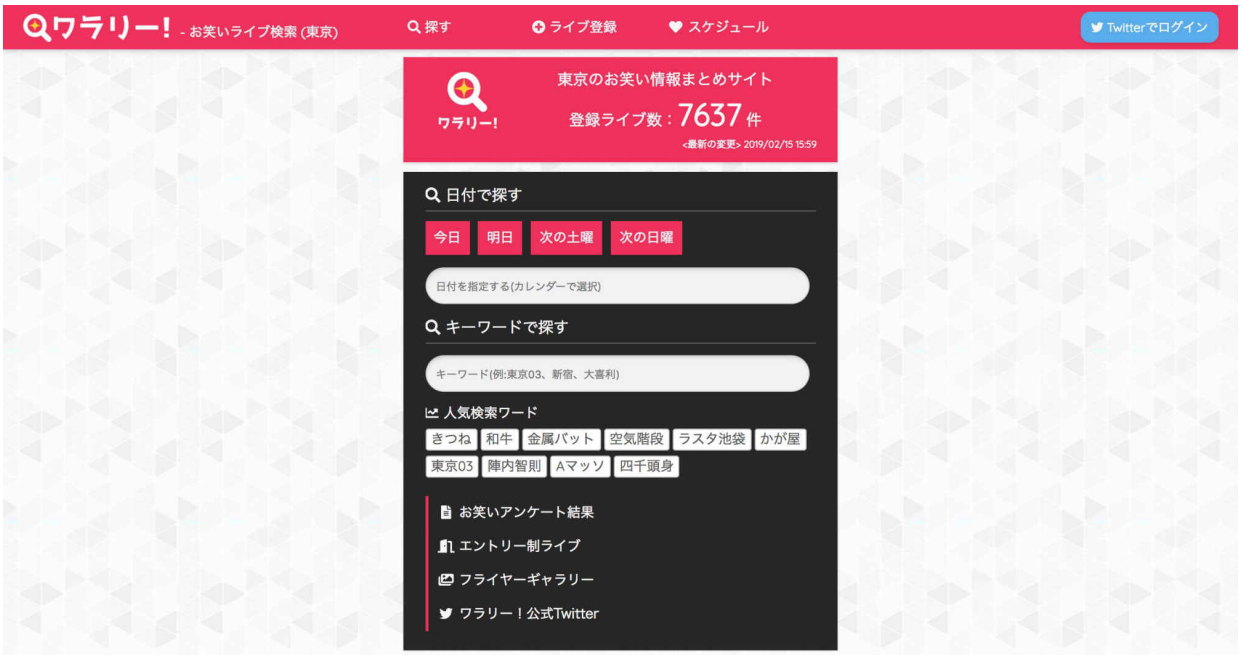
印象をよくするために、今風のきれいなデザインにするのは大事です。一方でデザインに凝ると時間がかかるのも事実です。なるべくカンタンにイケてる画面を作りましょう。

・デザインは紙に書く：ツールはたくさんありますが、紙とペンで十分です。初心者はデザインツールの操作よりも開発技術の学習を重視しましょう。

・1カラムにする：ワラリー！は8割がスマホからの閲覧です。PCとスマホで表示を変えるのは大変なので1カラムにしました。大手サービスの構造を真似すると良い感じに見えます。

・イメージカラーを決める：ワラリー！のイメージカラーはピンクです。旧サイトのイメージカラーが水色だったため、「旧サイトと差別化をはかりたい」「目立ちたい」「わくわくする感じにしたい」という意図から、寒色ではなく暖色を用いました。ヘッダやサイト内のボタンなど、アクセントカラーとして使用しています。

図1.3: トップページ


ワラリー! - お笑いライブ検索 (東京)
探す
ライブ登録
スケジュール
Twitterでログイン
ワラリー!
東京のお笑い情報まとめサイト
登録ライブ数：7637 件
<最新の変更> 2019/02/15 15:59
日付で探す
今日
明日
次の土曜
次の日曜
日付を指定する(カレンダーで選択)
キーワードで探す
キーワード(例:東京03、新宿、大喜利)
人気検索ワード
きつね
和牛
金属バット
空気階段
ラスタ池袋
かが屋
東京03
陣内智則
Aマッソ
四千頭身
お笑いアンケート結果
エントリー制ライブ
フライヤーギャラリー
ワラリー！公式Twitter

## 1.2　2か月目 エラーで行き詰まってもくもく会に参加しまくる

**機能を決める**

　サービスを作り始めるときは、「あんなこといいな、できたらいいな」とやりたいことがどんどん浮かんできます。まずは全部書き出しましょう！ 思いつくものを羅列できたら、難易度や優先度を振り分け、リリースまでに開発する機能を絞ります。

**コードを書く**

　書かなければサービスは動きません！いよいよ書くべし！書くべし！！です！！！

**モチベーションを維持するコツ**

　わからないことや解決できないエラーにつまづきます。そんなときは「もくもく会」などのエンジニアが集まるイベントに行って相談しましょう！「自分のサービスを作っています」と言うと、快く相談にのってもらえます。オンラインのQ&Aサービスや SNSで質問することもできますが、初心者だとエラー内容を適切に質問するのは難しく、コミュニケーションに疲弊します。慣れないうちは対面での質問をおすすめします。

**サービスを完成させるコツ**

**ちょっとずつでいいので開発し続ける**

　期間が空くと「どこまでやったっけ？」と状況を忘れてしまいます。なるべく毎日ちょっとずつ開発しましょう。また「やりたいことリスト」でタスクを忘れないようにします。進捗が遅すぎて「完成するのか？」と不安になるかもしれませんが、続けていればいつかは完成します。

**タスクは細かく分解する**

　GitHubのIssue機能を使って管理しましょう。「ユーザーの登録機能」と一口にいっても「メール認証」「SNSログイン（Twitter、Instagram、Facebook、Google）」「退会した場合、何週間まで復旧できるようにするか」など、細かい仕様はサービスによって異なります。メソッド単位までタスクを分解することで、進捗を可視化し、モチベーションを上げることができます。

**ツールや言語はあれこれ手を出しすぎない**

　初心者の場合、ただでさえ技術を学ぶのが大変です。なるべく絞りましょう。ワラリー！は、下記の言語で構成されています。

・マークアップ言語（HTML/CSS/SCSS）
・バックエンドの開発言語（Ruby/Ruby on Rails）
・フロントエンドの開発言語（JavaScript/jQuery）

　新しく覚えて使用したツールは下記です。

・Homebrew：Macのパッケージ管理ツール。ツールのインストールに使います。
・Visual Studio Code：IDE（統合開発環境）兼エディター。ソースを書きつつターミナルでテストできるのが利点。以前はSublime Textの無料版を使いましたが、ポップアップ広告で集中しづらかったです。VS Codeは無料で、広告も出ません。
・PG Commander：PostgreSQLを操作するGUIツール。RailsアプリをHerokuで使うには、初期設定のSQlite3からデータベースを変更する必要があります。
・GitHub：ソースコード管理サービス。様々なサービスと連携できる。
・Slack：コミュニケーションツール。もくもく会やエンジニアコミュニティに所属すると、質問・相談を行えます。
・Heroku：カンタンにRailsのサービスを公開できます。ただ、無料枠はすぐに制限がかかってしまいます。ワラリー！では、hobby dev（月額$7、24時間稼働）とPostgres Premium（月額$9、1000万レコードまで使用可能）に課金しています。

学習を諦めたけど、結果としてなんとかなったフレームワークやツールもあります。

・AngularJS：JavaScriptのフレームワーク。「このカレンダーのデザインいいな」と思った技術がAngularでしたが、jQueryで代替しました。
・Bootstrap：とても有名なCSSのフレームワーク。Bootstrapのテンプレートを使いたかったのですが、しっくりくるデザインがなく、結局CSSを手書きしました。
・Bourbon：CSSのフレームワーク。ライブラリーを導入した際に依存関係があったらしくエラーが出て、結局そのライブラリーごと使うのを止めました。
・Masonry：タイル状に可変で記事を配置できるRailsのGem。パソコンでは3カラムのタイル状レイアウト、スマホでは1カラムのレイアウトにしたかったのですが、力尽きて結局パソコン・スマホ共通で1カラムにしました。
・Prott：アプリ向けのプロトタイピングツール。操作を覚える時間がなかったため断念。ラフやデモは個人開発であれば省いた方が開発期間の短縮になります。
・Haml：Rubyを書く際に「erb方式でなくhaml方式で書いた方が実務に近い！」とアドバイスを受けたのですが、Progateなど初心者向けのRailsの教材はerbで書かれているので結局そのままerbを使いました。

## 1.3　3か月目 70%の完成度でリリースする

### フォームの入力テスト

1．まずは表示されるか、機能するかどうか
2．実際に投稿される想定のデータを入れて、レイアウトが崩れないか
3．悪意のあるデータや特殊記号を入れても問題が起こらないか（SQLインジェクション、クロスサイトスクリプティング対策など）

### 404エラー、503エラーなどのエラー画面の設定

Railsの標準だと英語のエラー画面が出てしまいますので、日本語の画面を設定しましょう。「Better Error Pages」というサイトで、3種類のデザインから選んでエラーページを作ることができ便利です。

図1.4: エラーページ


### ステージング環境をつくる

本番環境とは別に、サーバーでの動作を確認するためのステージング環境を作りましょう。Herokuの場合は、Heroku Pipelineという機能が用意されています。

### β版を公開してアドバイスをもらう

β版を友人に見てもらいましょう。サービスの根幹にあまり重要でない機能を提案されることもあるので、もらった意見を取捨選択しつつフィードバックを反映します。

**利用規約・プライバシーポリシーの作成**

利用規約やプライバシーポリシーが丁寧だと、ユーザーからの信頼度が上がります。『良いウェブサービスを支える「利用規約」の作り方』という本がおすすめです。

**サービスのリリースをお知らせする**

おめでとうございます！サービスがリリースできました！この時点であなたはめちゃめちゃすごいです。自分で自分を褒めてあげましょう。SNSで宣伝しましょう。

**Twitterで宣伝する**

リリース時はご祝儀訪問がもらえるチャンスです。アピール文言を練り、リンクを文中に仕込み、画像を添付しましょう。『朝に投稿 → 昼にRT → 夜に「RTありがとうございます！」と引用RT』の流れで告知効果が高まります。

「ワラリー！」では、相手によって文言を変えました。エンジニア向けには「初心者の成功体験」「100日間という数字」をアピール。「91RT、355いいね」いただきました。

> Web系未経験の初心者が100日間で初めてのrailsサービスをリリース
>
> https://warally.info
>
> 首都圏のお笑いライブを日付やキーワードで横断検索できるサービスです！
>
> ・5秒で使えるTwitterログイン
>
> ・わかりやすいUI
>
> ・13サイトからデータを毎日スクレイピング
>
> よかったら見てください～ （添付画像3枚）

お笑いファンには「できること」を強調して伝えました。「お笑いライブ検索サイトの閉鎖」から2週間というタイミングもあり、224RT、325いいねとかなり拡散されました。

> 【お知らせ】お笑いライブを検索できるサイトができました！
>
> http://warally.info/
>
> ・日付検索、キーワード検索ができます
>
> ・フライヤー表示機能あり
>
> ・誰でもライブを登録可能
>
> 今日やってるライブあるかな？という時にぜひご利用ください （添付画像4枚）

### Qiitaで記事を書く

サービス公開と同時に記事を書くと、Qiitaユーザーにもサービスを知ってもらうことができます。宣伝目的だとQiitaの規約違反となってしまうため、サービスで使用した技術の紹介をメインで書くと良いでしょう。

### Google Analyticsを使用して人気機能を見極める

どのページにアクセスが集まっているか、どこから集客できているかを分析するにはGoogle Analyticsを導入します。コンテンツの詳細ページを、posts/[:id]ではなくposts/[:title]にすると、GAの画面で人気記事がわかるので便利です。

### Google Search ConsoleでSEOを強化してアクセス数を増やす

SEO（Googleからの検索最適化）に力を入れると長期的なサービスの発展が望めます。

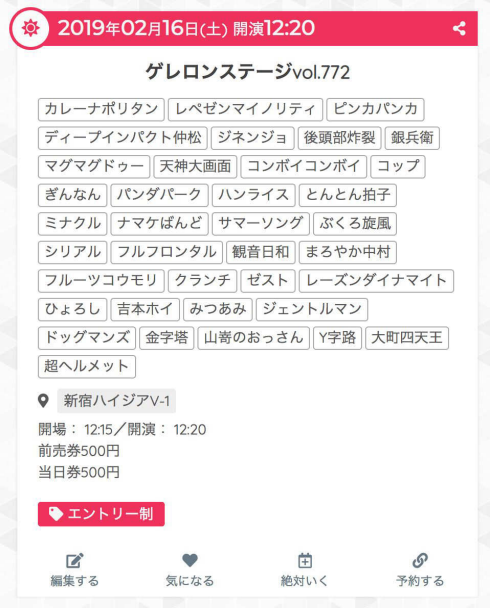
・サイトマップの生成とサイトマップの自動更新機能を実装する

・メタキーワードを設定する：ワラリー！の場合は「お笑いライブ」「チケット」「東京」「スケジュール」などを設定しました。

・ページタイトルを動的に設定する：「サイト名 | ページタイトル」ではなく「ページタイトル | サイト名」の順番で設定するとよいそうです。

・サイト内リンクを増やす：ワラリー！では日付、タイトル、会場、出演者などそれぞれの項目がリンクになっています。SNSシェアボタンも付けましょう。

図1.5: イベント個別ページ




### Google Adsenseによるマネタイズのコツ

Webサービスで稼ぐ場合、Google Adsenseによる広告を導入するのがカンタンです。ワラリー！では、月間5万PVに対して5000円程度の収益が発生しています。

・広告サイズはレスポンシブにする：サイズが大きいほど、クリック数が増えますし、閲覧によるインプレッション収益が上がります。ワラリー！では、最初は控えめにラージモバイルバナー（320×100サイズ）を使いました。しかし、売上が伸びず赤字になったため、レスポンシブ（336 x 280サイズ）に変更しました。フィード型のサービスなので、ファーストビューには広告を置かず、5つのライブにつき1つレスポンシブ広告を表示します。広告サイズを変えたことで、11月に500円だった収益が12月には5000円と10倍になりました。

・不快な広告はブロックする：毎日使ってもらえるサービスにしたいので「デリケートなカテゴリー」（出会い系、宗教、消費者金融など）や「マンガ系の広告主」（エロ・グロ系のマンガ）をブロックしました。これらの広告はクリック単価が高いという側面もあるので、サービスのテーマや世界観に応じて、ブロックするカテゴリや広告主を決めると良いでしょう。

## 1.4　まとめ

　初めて個人開発を始めたときは「一生完成しないんじゃないのか」と不安な毎日を過ごしていました。しかし、諦めずにいろんな方に相談して助けてもらったことで、なんとかリリースまでこぎつけることができました。

　これから始めるかたは「こんなに大変なの？」と心が折れそうになる瞬間もあるかもしれません。そんなとき、この本がモチベーションにつながるよう、祈っております。

```
かしい


フリーの駆け出しWEBエンジニア。お笑いライブの検索サイト「ワラリー！」を運営中。
もともと外資のITコンサル会社で業務システム設計をしていたが、独学でRuby on Rails、
JavaScript などを勉強して独立。技術ブログでは初心者向けの情報を発信中。


https://warally.info/
```

# 第2章　とがったテーマの婚活サイトを開発して得た経験

　高野と申します。僕は個人で「MatchLab」[1]という婚活マッチングサービスを開発しています。「本棚を覗けばその人が分かる」がコンセプトで、読書好きな真面目なユーザーに受けています。今回はこのサービスの開発経緯や良かったこと、難しかったことについて書きたいと思います。

図2.1: MatchKab

## 2.1　作った動機

　個人で婚活サイトを作ったという話をすると、必ずと言って良いほど「なぜ作ろうと思ったのか？」と聞かれます。答えは簡単。単純に彼女が欲しかったからです。

　開発を始めたのは、ちょうどマッチングアプリが市民権を獲得し始めていた時期でした。TinderやPairsといったサービスはありましたが、どうにも自分の求めている雰囲気とは違いました。自分好みの真面目な女性が登録してくれそうなサービスがどこにもない。

　それならばということで、自分で作ってしまおうと思ったわけです。何より「個人で出会い系サイトを作っちゃいました」という自己紹介ができたら面白いですよね。ということで、開発を始めました。

　ちなみに、初期は「学術や社会問題を議論できる恋人と出会えるサービス」という尖ったコンセプトを掲げていました。

## 2.2　開発中に苦労したこと①：技術レベルが足りない！

当時の僕は人材系の会社で営業として働いていました。プログラミング経験は初心者レベルです。

TECH::CAMPの1週間コースで初めてRuby on Railsを学んでから3ヶ月程度。仕事から帰った後の自宅や週末に勉強を続けていました。一通り基本的なCRUDやjQueryによる簡単なDOM操作、Herokuへのデプロイまではできるくらいの状態です。サービスをゼロから作るにはまだまだ足りないものばかりでした。

ひとつひとつ調べながら地道に実装していくことになりました。「こういう機能が欲しいよね」と思いついたら、まずQiita等で実装方法を調べる。ひとつの記事を読んだだけでは大抵は分からない。同じ機能について説明している記事をいくつか読んで、微妙な違いを見比べながら自分の手で実装していきます。

苦労はしましたが、成長実感が強く、楽しさの方が勝りました。ひとつ機能を実現する度に、自分の中で実装パターンが増えるのを実感できました。

## 2.3　開発中に苦労したこと②：デザインができない！

開発は勉強すれば楽しく進められたのですが、困ったのはデザインでした。知見はゼロ。センスもゼロ。作れば作るほどダサい画面が量産されていきました。

開発中の画面のスクショをTwitterに上げて友達に意見を求めたり、他サービスの配置を参考にしたりしたものの、大幅な改善はできませんでした。

最終的には、見栄えの良さは諦めて、ダサい画面のままでリリースしました。とにかく形あるものとして世の中に出すことを優先しました。

ところが、デザインが多少ダサくても、一定のユーザーは使ってくれました。サービスのコンセプトが尖っていたためでしょうか。「Done is better than perfect.」（完璧を目指すよりまずリリースせよ）という言葉が身に沁みました。

## 2.4　開発中に苦労したこと③：リリースに踏み切れない！

最後に苦労したのはリリース直前でした。最低限の機能だけ実装して早く世の中に送り出すことが大事だ。頭では分かっているのですが、これが非常に難しい。僕にとっては2つの障壁がありました。

ひとつ目の障壁は「不安感」でした。最初に色んな機能を同時にリリースしたほうが良いのではないか。機能が不十分な状態だと、初期ユーザーが「こんなもんか」と失望して去ってしまうのではないか。二度と戻ってこないのではないか。

後から考えると不要だった機能を詰め込んでしまい、時間を無駄にしてしまいました。お相手とのディスカッション機能やイベント機能などをリリース前に実装してしまったのです。

リリース直前になると「あったら嬉しい」要素が次々と思い浮かびました。トップページの情報を充実させた方が良いのではないか。初回ログイン時に使い方を説明するモーダルを出した方が良いのではないか。

今にして思うと、不安に思う必要はありませんでした。むしろ、あえて初期の機能を少なめに制限することもアリだなと思っています。初期リリース時に機能を少なめにしておく。リリース後に徐々に機能を足していく。そのたびにプレスリリースを打つ。すると、サービスを露出する口実が増えます。継続的に機能が追加されていることが世の中に伝われば、その分だけサービスの信頼度も高まります。

2つ目は「リリース作業の意外な多さ」でした。リリース直前になって最終確認するとボロボロと「これも必要だった」という作業が見つかります。僕がMatchLabをリリースするときには、以下のような作業を行いました。

・Google Analyticsタグの埋め込み
・OGP設定
・利用規約・プライバシーポリシーの準備
・退会機能
・お問い合わせ窓口の設置

後から考えると必須の要素ですが、ギリギリまで抜け漏れていました。

ひとつひとつは小さいものの「やるしかない」タスクです。気合いでやりきりました。リリースまでのチェック項目リストがあったらもっと心の準備ができたのかもなあと思いました。

## 2.5　開発中の良かったこと

苦労したことを先に述べましたが、実際には作ってみて良かったなと思うことの方が多くありました。その中でも最たるものはやはり「勉強になる」という一点です。

自分でゼロからサービスを作る場合、HTML・CSS・jQueryからサーバーサイド、インフラまで、全て自分で構築しなければなりません。必要性に迫られて強制的に勉強できました。

開発を進めれば進めるほど、自然と自分の中に「良いサービスにしたい」というモチベーションが湧きました。このモチベーションが学習意欲をドライブしてくれました。非常に良かったです。

また、システムの実装だけでなく、SEOやマーケティング、デザイン等まで勉強する機会にもなりました。小さな枠に収まらない幅広い経験を得ることができました。自分の武器が増えたことを実感しています。

## 2.6　リリース後の大変だったこと

様々な苦労を乗り越えて、ついにサービスをリリースしたわけですが、個人サービスはここからが本番です。リリース後にも様々な困難に直面しました。

最初に大変だったのは、メッセージ機能に重大なバグがあったことです。なんとメッセージ一覧画面のクエリが間違っていて、他のユーザーが異性を口説いているメッセージが表示されてしまっていたのです。

リリース直後に気付いてすぐに修正しましたが、これには冷や汗をかきました。ずっと「自分」と「異性ユーザー」だけの状態で動作確認していました。複数ユーザー同士が交流する状態のテストが漏れていたのです。

それ以外にも、開発後のタスクが意外と多いことに苦労しました。特に宣伝活動、SNS広告の運用やプレスリリースの作成、お問い合わせ対応などには時間を取られました。

できるだけ1人で楽に運営するため、エラー通知やお問い合わせは全てSlackに通知を集約してリアルタイムに把握する、ボタンひとつでメールマガジンを配信できるように自動化する、等の工夫を試しています。

## 2.7　リリース後の嬉しかったこと

大変なことも色々ありましたが、嬉しかったことの方が沢山あります。最大の喜びは、ユーザーからの反響です。

このサービスを公開した時には初日だけで100名近くの新規登録がありました。実際にユーザーが使ってくれていることに深い達成感を味わいました。

その後、ピボットやリニューアルを経ながら現在まで約3年が経過しました。ユーザー数は2,000名を突破しました！

サービスを通して恋人を見つけて付き合い始めたという報告も、僕が耳にしているだけで10組以上に上ります。さらには最近、その中で結婚にまで至ったカップルが2組、うち1組は子どもができたという話まで聞いています。

自分が作ったサービスを通して、人が幸せを見つける、新たな命が生まれる。こういった経験はなかなか得られるものではありません。非常に良い経験だったなと思います。

それ以外にも、転職活動でサービスとソースコードを見せながら苦労話や工夫した点などを話すと評価されやすかったり、初対面の人にも「出会い系サービスを作ってる高野くんね」と覚えてもらいやすかったりと、良いこと尽くしです。

## 2.8　最後に

　個人開発の良いところは、誰に文句を言われても自分の信じるものを作り続けられることだなと思います。

　MatchLabを開発する中で、何度も「恋愛マッチングではなく純粋に学術を議論できる友達を作れるサービスにすれば良いじゃん」「読書仲間を見つけられるSNSにすれば良いじゃん」といったことを言われました。

　でも正直、そんな普通なサイトを作っても、つまらないじゃん！と思います。他人に何を言われても、個人が信じる方向に勝手に爆走できる。それが楽しい。

　僕はプログラミングの学習というのは、基本的には難しい、辛いものだと思うんです。でも「作りたいと思えるサービス」に出会えれば、その辛さを乗り越えられると思うんですね。

> 「こういうサービスが本当に欲しい、使いたい、開発者特権で無料で使い続けたい」

　そんな情熱を持てる独創的なサービスをぜひ探してみてください。

```
H.高野


慶應義塾大学商学部を卒業後、人材系企業で営業を経験。
現在はWEB企業でエンジニアとして働くかたわら「本棚を覗けばその人が分かる。」
読書好きのためのマッチングサービス「MatchLab」を運営している。


match-lab.com
```

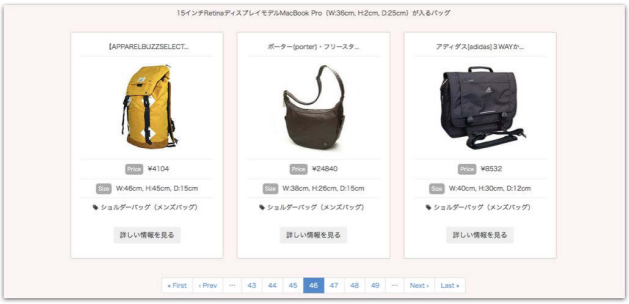
1. https://match-lab.com/

# 第3章　アイデアからリリースまで、Webサービス開発を徹底詳説

　はじめまして、ゆずたそ（@yuzutas0）と申します。『10個のWEBサービスを開発した』『1日でサービスをリリースした』などの記事を公開しています。今回は「HileSearch」（ハイルサーチ）開発エピソードを通して、サービス開発の流れを詳しくお伝えします。

## 3.1　サービス紹介 - ありそうでなかったWEBサイト

図3.1: HileSearchの画面


　持ち運びたいノートPCを選ぶと、そのPCより大きいバッグを一覧表示するサービスです。「PCが入る（ハイル）バッグを検索（サーチ）する」という名前です。

サービスの使い方

図3.2: HileSearchの利用の流れ


　①サイトにアクセスしてPCを選択します。持ち歩きたいPCのブランド（例：Mac）を選ぶと、候補が表示されるので、機種名（例：MacBookPro Retina 15inch）を選択します。該当のPCを持ち運べるバッグが一覧表示されます。

　②条件で絞り込みます。例えば「メンズのショルダーバッグ」（タグ）や「価格が安い順」（並び方）を選べます。条件設定後に検索ボタンを押下すると結果が表示されます。

　③好みのバッグを購入します。「詳しい情報を見る」を押すと、Amazonの商品ページに遷移します。店舗で実物を確認したり、そのままAmazonで購入しましょう。

## 3.2　企画 - ユーザーと向き合って考える

きっかけはメンバーの困りごとから

　もともとは私のアイデアではなく「新規事業を作ろう」という有志の集まりで考えました。「WEBサービスは誰かの困りごとを解決するものだ」と考え、各メンバーが抱えている悩みを持ち寄りました。メンバーの1人から「買いたいバッグがあるのだけどノートPCが入るか一目で分からなくて不便だ」という話が上がりました。

図3.3: サービス企画の仮説ポイント


　企画の初期で重視したのは「誰の」「どんな課題を」「どう解決するか」です。[1]最初の仮説（バージョン1）は以下のように整理しました。

- Customer: サイバーエージェントやリクルートで仕事をガツガツ進めるような女性をペルソナに想定。ノートPCを頻繁に持ち運ぶが、可愛い鞄を好むので「見栄えの良さ」と「PCを持ち運べること」を両立したいと考える（はずだ）。
- Problem: 欲しいバッグにPCが入るか調べるのが大変だと感じている（はずだ）。PCを持たずに店員さんに質問しても「A4サイズなら入ります」「そのPCが入るかは分かりかねます」という情報しか手に入らない（らしい）。
- Solution: バッグとPCを入力 → PCがバッグに入るか分かるスマホアプリ（案）。

インタビューを通して仮説を検証しよう

　あくまでも仮説です。正しいかは分かりません。プログラミングの前にインタビューから始めました。アプリの画面遷移（ワイヤーフレーム）を画用紙にスケッチして、その画用紙をアプリに見立て、アプリを使うシチュエーションを想像してもらうのです。

　ロールプレイの過程で「どのような時に」「どのような場所で」「どのような気持ちで」「どのような手順で」「どのくらいの価格の」「どのようなバッグ」を買っているのか聞きます。ユーザーがバッグを買うときの5W1Hを理解することが目的です。

　老若男女問わず20人にヒアリングすると、仮説が外れていたと分かりました。

図3.4: インタビューで分かったキラキラ女子の実態

| タテマエ | | ホンネ |
| --- | --- | --- |
| このバッグを買っていいのか悩む〜〜<br>PCが入らないと困る〜〜 | VS | 絶対にこのバッグは買う<br>たとえPCが入らなくても買う |

IT系イケイケ女子は、欲しいと思ったバッグは絶対に買うのです。口では「PCが入らないと困る」と言っていたのに、実際の行動は「PCが入らなくても買う」のです。わざわざアプリで「PCが入るかどうか」を調べるモチベーションはないと分かりました。

インタビューで気付いた「別の可能性」

PCを持ち歩くのは営業職、就活生、IT企業のプランナー、フリーランスのエンジニアでした。中には「Amazonでサイズを見比べてバッグを買ったことがある」と答えた人もいました。追加ヒアリングを行い、新しい仮説（バージョン2）を立てました。

- Customer: 服装自由のWEB企業に勤める人、もしくはフリーランス。よくノートPCを持ち歩く。カバンをネット通販で買うことに抵抗がない。
- Problem: バッグを探すのが大変。店頭だとマイPCが入るか絶妙なサイズが多い。ネット通販でサイズを見比べるのが手間。いわゆる「PC用バッグ」（ビジネスバッグ）を買えば確実だが、見た目のバリエーションが少ないと感じており、できれば一般的なバッグから好きなものを探したい。
- Solution: PC機種を入力 → 入るバッグを出力するWebアプリケーション。

最初の想定に固執せず、ヒアリング結果を踏まえて方向転換（ピボット）しました。

挫折と再スタート - たった1人での開発

すぐに次の問題が浮上しました。「大儲け」できないのです。新規事業を立ち上げる有志の集まりですから、目指すのはあくまでも投資を受けて急成長できるサービスです。課題を突破できず、アイデアはお蔵入りに。筆者自身もサービス案をしばらく忘れていました。

思い出したのは半年後。新しいPCを買ったときです。PCの持ち運びに適したカバンを探したくなりました。自分自身が「HileSearch」を求める状況になったのです。

一度は作ってみたいと思ったアイデア。モチベーションはあります。インタビューを通してコンセプトは固まっています。余計なことは考えず開発に専念できます。「今やるべきだ！」「自分一人だけでも実現したい！」と思いました。

もう1つのメリット - コンテンツを自動収集するための技術力

技術の勉強になるというメリットもあります。プログラミング初心者が最初に作る「いわゆる掲示板のようなWEBサービス」とHileSearchには、違いがあります。

掲示板サービスだと、ユーザーがコンテンツをC（作成）・R（参照）・U（更新）・D（削除）します。一方でHileSearchはPCやバッグの情報を自動収集するサービスです。HileSearchを作ることによって、バッチ処理やスクレイピングの開発を学べます。

スクレイピングができると、個人開発の可能性は広がります。サービスの立ち上げで難しいのが、最初のコンテンツ集めだからです。

・一人でも使いたくなるサービス設計にするか。

・自然と人を呼び込む仕組みを設計するか。

・外部コンテンツを何らかの方法で活用するか（営業や提携も含めて）。

外部コンテンツを利用する方法の1つがスクレイピングです。きちんとマナーを守った上で、正しいスクレイピングができると、強力な武器になります。

## 「自分ルール」でペースを維持する

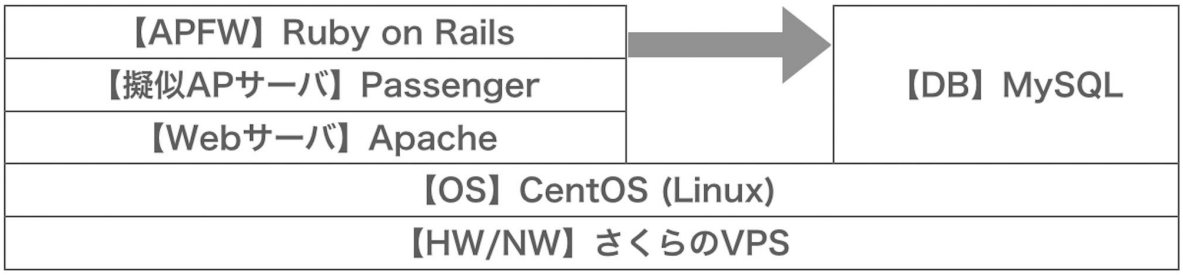
これだけメリットがあるのですから、迷うことはありません。ひたすら開発するだけです。モチベーションに左右されたくなかったので、自分ルールを設けました。「平日の仕事が終わったあと」「カフェが閉店になるまで」「居座って作業する」。

とにかくカフェに行き、PCを開く。「今日はやる気がない」「どうしようかな」という脳内議論は禁止。カフェまでPCを持ち運ぶのは、ちょっと地味なビジネスバッグ。サービスが完成したらお祝いにオシャレなバッグを買おう。さぁ、開発のはじまりだ！

## 3.3　開発 - 小さく始めるアグリゲーションサービス

個人の趣味サービスなので最小限のシステム構成とします。

図3.5: HileSearchのシステム構成

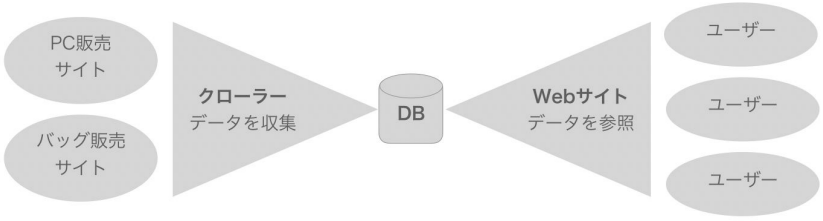



ハードウェアやネットワークは外部サービスに任せます。今回は、さくらインターネット社が提供するVPS（仮想専用サーバ）の安いプランを利用しました。

APFW（アプリケーションフレームワーク）はRuby on Railsです。「How to build a blog in 15 minutes with Rails」（15分でブログサービスを作る）というデモ動画で有名になりました。動画タイトルが示すように、WEBサービスのお手軽開発に向きます。

OS（オペレーションシステム）はCentOS、WebサーバはApache、DBはMySQLです。筆者が仕事で使ったことのある技術で安心感がありました。AP（アプリケーション）サーバはPassenger。ApacheでRailsを動かすためのモジュールです。

図3.6: Hilesearchのサービス構成



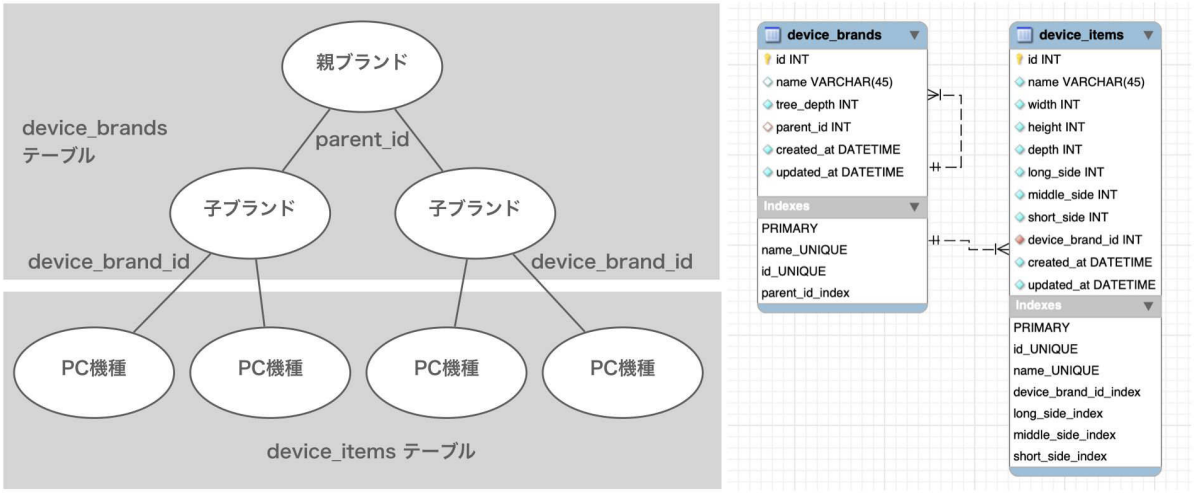

アグリゲーション型のサービスなので、2つの機能に分けて開発します。1つ目はインターネット上からPCやカバンの情報を収集するバッチ（クローラー）です。2つ目はユーザーがカバンの情報を検索・参照するためのWebサイトです。

データ収集バッチの流れ

PCデータの収集は「対象サイトのHTMLを取得」→「データを抽出・加工」→「DBに保存」という流れで行います。取得元サイトは慎重に選びます。規約でスクレイピングを禁止していないか。必要なデータを取得できるか。誤解を招かないよう、本書では対象サイト名は明記しません。

PCのテーブル（ER）を設計しよう

図3.7: HilesearchにおけるPCデータの構成




まずは画面に表示する「PC機種名」（nameカラム）が挙げられます。機種名をもとにユーザーは自分のPCを選びます。最終的に667種類のPCに対応しました。種類が多いので、そのままでは取り扱いが大変そうです。「機種名」（例: MacBook Pro）を束ねる「ブランド名」(例: Mac）も取得しました。

ブランド名はネスト構造を持ちます。「Let's Note」（親ブランド）の下に「RZシリーズ」（子ブランド）が存在します。機種（items）とブランド（brands）のテーブルは分けて、brandsは自己参照できる構造としました。子ブランドのレコードはparent_idで親ブランドのレコードを参照できます。

Railsで慣習的に使う「管理用id（id）」「データ生成日（created_at）」「データ更新日（updated_at）」カラムも各テーブルに設けます。

また、このサービスの特色として「サイズ」（縦・横・高さ）のデータを取り扱います。PCがバッグに入るかどうか計算するために必要だからです。

図3.8: 2つの立方体の3辺を比較してPCがバッグに入るかどうか判定する


横が長いバッグ、縦が長いバッグと、様々なバッグがあります。「計算用サイズ」もあると良いでしょう。縦・横・高さを小さい順番にソートして「short_side」「middle_side」「long_side」と名付けたカラムです。バッグの「short_side」「middle_side」「long_side」と比較することで、PCがバッグに入るのか簡単に判定できます。

### バッグのテーブル（ER）を設計しよう

PCと同様にしてバッグのデータ構成も設計します。画面に表示する「商品名」「写真」は必須です。クリックしたら購入サイトに飛べるように「URL」も欲しいですね。検索しやすいように「タグ」（例: メンズバッグ）も取得しました。PCが入るかどうか計算するための「サイズ」が記載されている「説明文」も必要です。

最終的には約10,000のバッグに対応したのですが「URL」はクローラー開発にも役立ちました。同じURLのデータがすでにあれば処理をスキップ（あるいは上書き）できるため、コンテンツが重複しないで済むのです。また、スクレイピングが失敗した時には、開発者本人が直接ブラウザでURLを開けば、簡単に原因を調査することができます。

```
# 多様なキーワードに対応
width = get_size ["横", "ヨコ", "長", "幅", "W", "W"]
height = get_size ["縦", "タテ", "高", "厚", "マチ", "マッチ", "まち", "H", "H"]
depth = get_size ["奥", "マチ", "マッチ", "まち", "幅", "厚","D", "D"]
```

取得元サイトではバッグのサイズの書き方が統一されていませんでした。商品の説明文（テキスト情報）から「おそらくこれがサイズだろう」と思える表記を抽出する必要があります。正規表現と分岐処理を組み合わせ、様々なパターンに対応しました。

### 小さいサイクルで設計・実装・テストを高速に回す

クローラー開発では、小さくコードを書き、小さく試し、エラーはすぐに直す、という検証を繰り返しました。3つの例を挙げます。

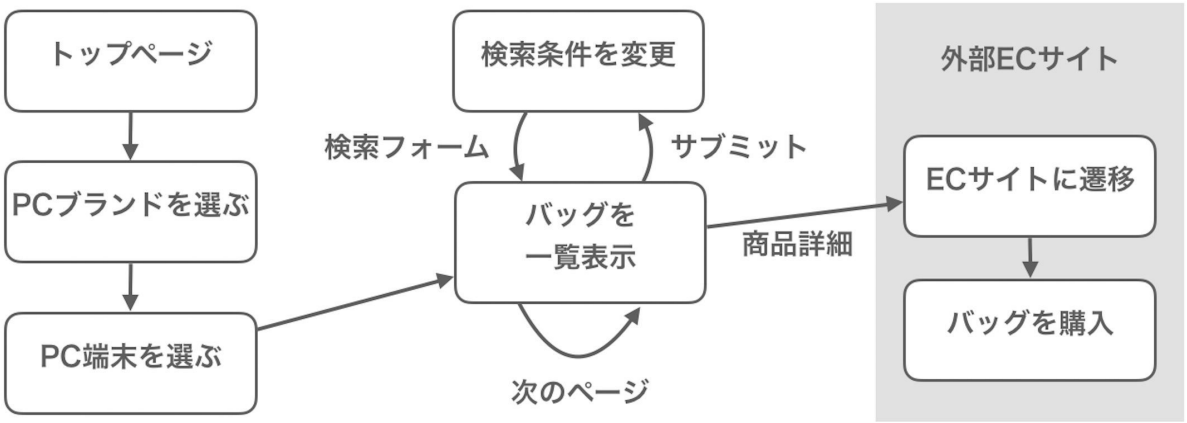
①バッグのサイズ（縦・横・高さ）を抽出する処理です。最初から完璧なアルゴリズムは作れません。エラーログを確認し、コードを修正して、また試して、またエラーログを確認する。この繰り返しで、精度を少しずつ上げます。

②再実行（リトライ）やスキップの処理です。アクセスエラーが起きるまでは実装しませんでした。「作ったけど使わなかった」「最初の作り込みに時間を掛けた割には進んでいない」といった個人開発のアンチパターンを避けるためです。

③ソースコードのクラス設計やメソッド設計です。Controllerクラスに何を書くか。Modelクラスに何を書くか。システム設計も同じです。最初は深く考えずに作ります。設計・実装・テストを繰り返すうちに「確認や修正に時間が掛かる」「あの処理はどこに書いたのかすぐ思い出せない」と感じる時が来ます。そのタイミングで、メソッドやクラスを分割して、シンプルで分かりやすい設計へとリファクタリングしました。

検索WEBサービスの開発

図3.9: ユースケースと画面遷移




バッチ（クローラー）を作ったら、次はユーザー向け画面の開発です。ユースケースを整理しながら画面遷移を設計します。ユーザーの一連の行動を想像して、漏れがないように仕様を洗い出します。

この画面遷移・機能一覧をもとに、ワイヤーフレームを紙に書き出し、ゴリゴリと画面を実装します。具体例として「検索結果の表示」について「画面イメージ」と「サンプルコード」を以下に示します。見比べると必要な項目を生成できていることが分かります。

図3.10: 検索結果の画面イメージ


```
class BagItemsController < ApplicationController
  def index  # HTTPリクエストを受けて画面を生成
    @validated_params = validates(params)  # 検索条件
    @device = find_target(@validated_params[:device_id])  # 端末情報
    @bags = search(@validated_params, @device)  # バッグの一覧
    @tags = tag_list  # 検索フォーム表示用
  end

  private
    def validates(params)
      # 検索パラメータに不正な値がないかチェックする
      # 不正な値がある もしくは 条件未入力 ならデフォルトの値を入れる
    end

    def find_target(device_id)
      # DBからPC端末の情報を取得する
      # PC端末が未指定 もしくは 存在しないid ならサイズ検索できないのでトップに遷移
    end

    def search(params, device)
      # 検索条件とPC端末サイズをもとにクエリを組み立てる
```

```
        # DBに検索クエリを発行して結果を取得する
      end


      def tag_list
        # タグの一覧をDBから取得する（例：メンズバッグ、ショルダーバッグ）
      end
  end
```

小さくテストしながら画面を作り込む

ワイヤーフレームはあくまでも初期構想です。実装中の画面を友人に見せて、アドバイスをもとに軌道修正しました。「ダサい」程度の意見であれば対応の優先度を低くして「遷移が分かりにくい」「手が止まる」といった反応は優先度を高めました。ユーザーの反応を元に開発する。だけどユーザーに振り回されない。この思想で画面を作ります。

実装後はブラウザでテストを行い、バグやデザインを修正しました。検索条件の組み合わせが複雑になるため、フォーム・ルーティング・検索クエリを重点的にテストしました。

デザインはお手軽です。ワイヤーフレームに似たサイトを探してCSSの参考にしました。CSSはTwitter Bootstrapを、アイコンはFont Awesomeを利用しています。

検索して表示するだけのサイトなので、画面の開発は簡単です。ユーザー登録の処理さえありません。将来的に「閲覧履歴」「お気に入り」「シェア」機能を作る場合も、ユーザー登録なしでセッション情報をもとに自動で履歴を管理すればいいでしょう。改めて「個人開発にうってつけなネタだなぁ」と思います。

## 3.4　リリース - 価値を届ける

本番サーバを構築しよう

　WEBサービスを公開するために、さくらのVPSでサーバを立てます。主に①「Linuxサーバの初期設定」（ユーザー権限やアクセス管理）→ ②「Webサービスを動かすために必要なミドルウェアを用意」（RailsやMySQL）→ ③「外部からRailsアプリケーションにアクセスできるように設定」（Apache）といった流れになります。

　この手の作業はスムーズに行かないものです。筆者はデータ移行やコンパイル周りでトラブルシューティングに苦戦しました。

図3.11: リリースのために実施する作業

| ① Linuxサーバの構築 | ② ミドルウェアの用意 | ③ サービスの公開 |
|---|---|---|
| ・VPSやドメインの申し込み＆設定<br>・yumパッケージの自動更新設定<br>・root以外のユーザー生成＆sudo権限付与<br>・SSH鍵認証の有効化<br>・SSHポート番号の変更<br>・SSH認証失敗時の自動ブロック設定<br>・ファイヤウォール設定<br>・ウィルス対策ソフトのインストール<br>・ファイル改竄検知の設定 | ・MySQLのインストール＆起動<br>・MySQLユーザー追加＆権限付与<br>・RailsのDB接続設定<br>・Railsランタイムをインストール<br>・HileSearchのソースコードを配備<br>・LinuxユーザーにRails実行権限を付与<br>・MySQLでマイグレーションを実施<br>・静的ファイルを本番向けにコンパイル | ・PassengerとApacheの依存ライブラリをインストール<br>・Apacheの設定をセキュア化<br>・過去サーバログの自動圧縮設定<br>・HTTPリクエストをRailsアプリケーションに当てる<br>・Apache + Passenger + Rails の起動 |

自分がユーザーとして使ってみる

　リリースしたらドッグフーディング[2]です。「実際にバッグを買うぞ」という気持ちで使うと、次々と改善点が見つかりました。

　例えば「（ショルダーバッグなど）タグが豊富だと探しやすい」と考えたのですが、実際はタグが多すぎて絞り込むのが大変でした。

　また「PCを入れるバッグなのだからPCで探すだろう」「インタビューによると高い買い物はPCでやる人が多いらしい」と見立てました。しかし、購入前の情報収集フェーズでは、むしろ通勤電車でモバイルを使って探したくなるものです。筆者自身はそうでした。残念ながらモバイルのレイアウトは考慮できていませんでした。

　他にも「本当に探しているときは、複数のバッグを比較したい（だけどHileSearchのデザインでは比較しにくい）」「本当に買うときは、購入者レビューを参考にしたい（だけどHileSearchにはレビュー情報がない）」といった問題が浮上しました。

出てきた課題は、修正が容易なものから順次改修しました。ちょっとした画面表示であればすぐに対応できます。モバイル対応やデータ項目追加は、後から直すと大変です。個人開発者として場数を踏むと、「最初に設計すべきこと」と「後から直せるもの」を見極めて、筋の良い企画・要件を考えられるようになります。

### ユーザーから寄せられた言葉

リリース直後にGIGAZINEというメディアに掲載いただき、[3]サービスが広まりました。Twitterやはてなブックマークのコメントを一部引用させていただきます。

> 「こういうの地味に便利ですよねえーー」
> 「ちょうど欲しいと思ってた！」
> 「この発想は…すごくいい！利用してみます」
> 「これはカバン好きな俺得アプリ！」
> 「大変助かる。リュックは電車で本を取り出せない」
> 「MBPが入る可愛いリュックを求めてるのでこういうのうれしい」
> 「最高じゃないですか・・・」
> 「カバン選びに苦労している自分にはHOT」

図3.12: HileSearchを紹介しているユーザー


名前も知らない人が自分のサービスを使ってくれている。接点のない人が自分のサービスを紹介してくれている。クリエイターとしてこれほど誇らしいことはありません。

最初のインタビューで紙のモックを見せたとき、魅力がないとダメ出しされたことを思い出しました。企画を実現して、企画の良さを認めてもらえることが、こんなに嬉しいとは思いもしませんでした。胸の奥から熱いものが込み上げてくるのを感じました。

また、開発の進め方をブログに書いたところ、ポジティブな言葉が寄せられました。

> 「Webサービス作成する上で参考になる」
> 「企画、ニーズ調査、要件定義、設計、開発、テスト、チューニング、リリース、一通りの流れを全て経験して、こうして共有されてるの良い」
> 「ちゃんと目標設定して課題解決しながら着実に前に進んでて尊敬する」
> 「1人で作れそうな規模感で収益化できそうなアイディア浮かぶのが羨ましい」
> 「こういう、自分がほしい機能とかなんでもつくれるエンジニアかっこいい」

かつて筆者自身が同じことを言っていました。いつかアイデアを自由自在に実現できるクリエイターになりたい。自分もこんな風にこんなサービスを作れるようになりたい。ほんの少し前まで、筆者自身が「憧れる側」でした。HileSearchを開発しようと決めてからは、がむしゃらに、ただ愚直に、ソースコードを書き続けただけです。いつの間にか「こうなりたい」と言ってもらえる側になっていました。少しずつではありますが、着実に、理想の自分に近づけたのだと嬉しく思いました。

「使いにくい！」は「ありがたい」

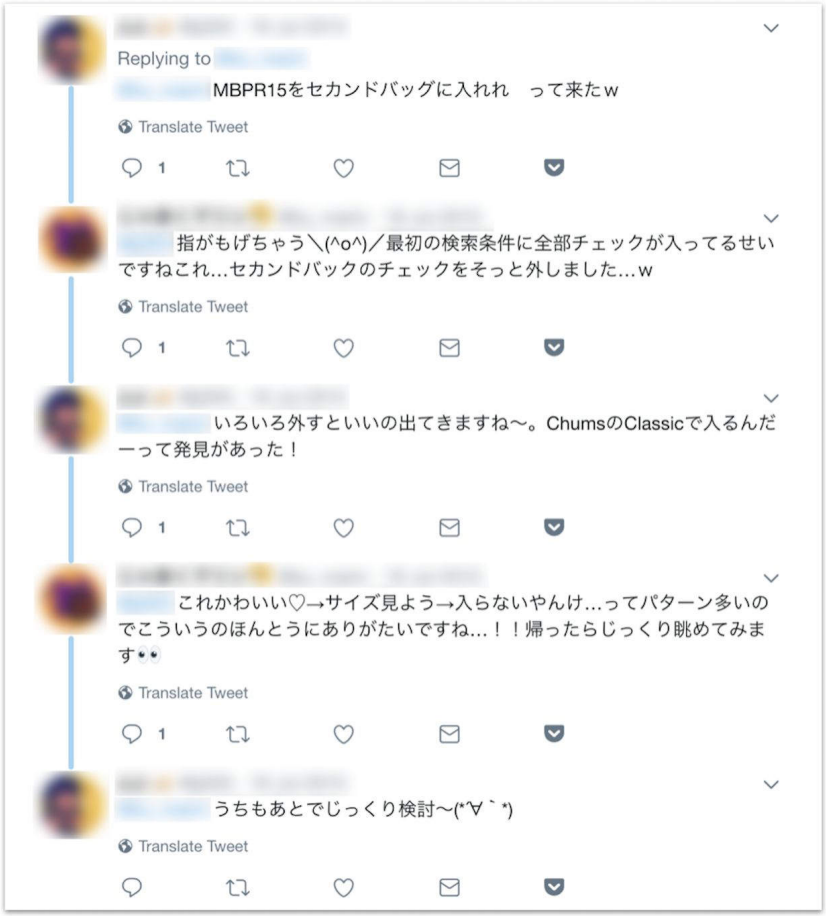
一方で、本当に悔しいくらい、ありがたい言葉の数々も寄せられました。

> 「ThinkPadX220が無いので解散」
> 「Surfaceシリーズがない(´；ω；`)」
> 「OR検索しかできない」
> 「チェックボックスの付け外しがクソメンドクサイ」

わざわざコメントしてくれたのは、コンセプトへの期待があったからだと思うのです。そして、もし本当に筆者が一流の企画者だったら、どれも防げる課題だったはずです。

代表例がPC端末の対応です。スクレイピングしやすいデータを集めて「600種類のPCに対応しました！」と掲げました。技術ありきの安易な発想だったと反省しています。人気のPC機種をリサーチして、手動でいいからデータを入れて、最大多数のユーザーのニーズに素早く応える。これが企画屋の発想です。当時の自分にはできませんでした。自分で企画をして、指摘を受けて、少しずつ身に付くのだと思います。

図3.13: 使い方を模索しているユーザー




UIの分かりにくさも反省点です。上のキャプチャでは、ユーザー同士が「これどうなっているの？わかりにくいね」「こうするとできるよ！」「ほんとうだ！」と教え合っています。これはこれで嬉しかったです。わざわざ試行錯誤してくれているのですから。

だからこそ、そこまでやってくれるユーザーに対して、使いにくいサービスしか提供できなかったことを悔しく思いました。また、それ以上に多くのユーザーは、期待を裏切られたと感じて離脱したはずです。

どんどん修正するしかない！！！

どのみち引き返せません。リリースしなければ修正点に気付くことさえできなかったでしょう。いつまでもアウトプットを世に出せない、自称クリエイターからは卒業したいですよね。だったら失敗を後悔するより、失敗に気付けたことを喜んで、対策前進するしかないですよね。ということで、ひたすら修正リリースです。実例を挙げます。

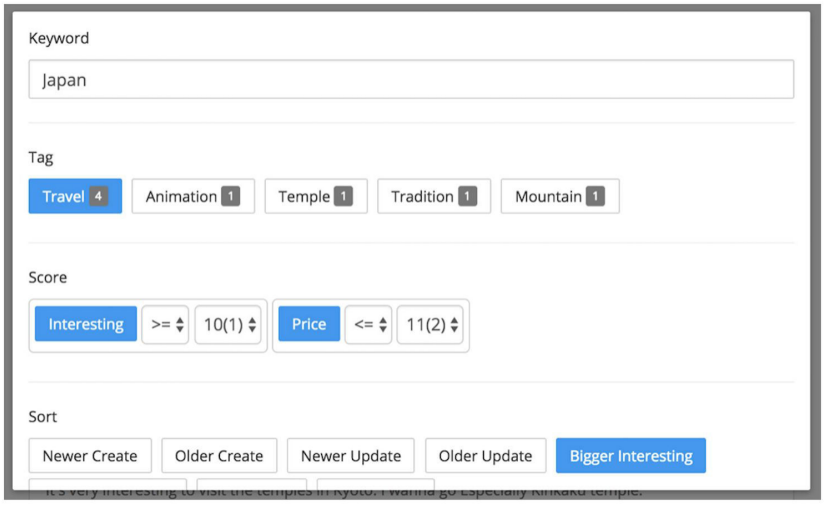
図3.14: HileSearchの修正画面1


「タグ検索」では、一括選択・一括解除ボタンを追加しました「タグが多すぎて付け外しが面倒臭い」という声が多かったからです。また「OR条件」だけでなく「AND条件」も選択可能にしました。検索機能は本当に難しいです。「ANDが当たり前だ」という人もいれば「ORが当たり前だ」という人もいます。かといって選択できるようにすると「ユーザーに選択させないのが良いUIだ」と怒られます。難しい。

図3.15: HileSearchの修正画面2


「ページネーション」も修正して、「次へ」ボタンをページ上部に表示しました。マウスカーソルを常に同じ場所（ページ上部）に固定したまま「次へ」ボタンを押すことができます。元々はページ下部にのみ「次へ」ボタンがあり、バッグ画像が大きい・小さいと、ボタンの位置がずれてしまいました。マウスカーソルを移動する必要があるので、ユーザーはバッグの比較検討に集中できません。

図3.16: もっと使いやすい検索フォームのUI - My100tales




色々と考えたり、修正したり、他のWEBサービスを見るうちに、良いUIデザインが徐々に分かってきました。筆者がHileSearchの後に作った「My100tales」（Qiita:Teamライクなドキュメント管理サービス）の記事検索フォームは格段に使いやすくなっています。実践するからこそ、勘所を掴めるのだと思います。

## 趣味開発で仕事の可能性が広がった！

企画・開発・運用と一連のプロセスを回したこと。ビジネスやシステム全体を見ながらベストな設計を模索したこと。個人開発の取り組みが評価されて、会社で開発チーム立ち上げを任されました。iOSアプリしか興味がない。デザインしか興味がない。そういう人が多く、全体を見て動ける人材は重宝されるようです。

ひとつひとつ着実にチャンスを掴むことで、案件が次々と舞い込みました。最近では大規模カンファレンス・300人の前で登壇して、ベストスピーカーにも選ばれました。

HileSearchの個人開発はキッカケの1つだったと思っています。自分が楽しいと思ったことを貫いて、きちんと行動して、やり抜いたことで、可能性が広がりました。

## できなかったこと

仕事は多忙を極め、HileSearchを更新できなくなりました。このWEBサービスにはまだまだ可能性がありました。

ひとつ目が、横展開による収益強化です。こんなコメントが寄せられました。

> 「いろんなバージョン欲しい。カメラバックとか。」

2つ目が、メディア化によるSEO・集客強化です。とあるファッション系ブロガーさんにこんな記事を書いていただきました。

> 「HileSearch（ハイルサーチ）という便利なサービスを使ってみました！」
> 【12インチの新MacBookが入るお洒落なクラッチバッグ7選】

どちらも実現は難しくありません。PC端末のサイズを取得したように、カメラなどの持ち運びたいアイテムのサイズを登録すればいいのです。ファッション好きなライターにHileSearchを使ってもらい、キュレーションコンテンツを書いてもらえばいいのです。仕事をしながらでも何かできたはずです。

それでも更新できなかったのは、システムの保守性が低かったからです。例えば、スクレイピングでバッグのサイズを抽出するための正規表現。本当ならテストコードを書いて、正規表現が期待通りに動作しているのか、自動チェック可能にすべきでした。筆者はテストコードを書かずに勢いで作ったのです。エラーログの自動通知も不十分でした。

仕組みを整備できなかったのは、筆者のスキル不足が原因です。ユーザーの皆さまに申し訳なく思っています。企画を120%生かし切れなかったことを悔しく思っています。

だからこそ、もっと良いものを作れるようになりたい、足りないものを学びたいと強く思います。あるいは苦手なところを補ってくれる人と一緒にやっていきたいと思います。

## 3.5　まとめ -「私の個人開発」奮闘記

「誰かと一緒にやる」と書きましたが、それは個人開発と呼ぶのだろうかと、筆者自身も分からなくなりました。もしかしたら、筆者にとっての個人開発というのは「その企画に関わる個人個人の思いを実現するもの」かもしれません。一人でもいいし、一人じゃなくてもいい。仕事でもいいし、趣味でもいい。儲かってもいいし、儲からなくてもいい。だからこそ「私の個人開発」は、いつも最高の挑戦であり続けるのだと思います。

最後まで読んでいただき、本当にありがとうございます。[4]

```
ゆずたそ / @yuzutas0


（自称）企画屋・コンセプトデザイナー。WEBに興味を持ったきっかけは、依頼を受けて
小学校の特別授業を企画したこと。グラミン銀行と提携してSkypeによるオンライン国際
交流を実現。子供達の笑顔が国境を超えた瞬間を目の当たりにして、ITの可能性に気付く。


https://hilesearch.yuzutas0.com
```

1. 体系的な仮説構築の方法論を知りたい人は「リーンキャンバス」で検索してください！
2. 自分で自分のサービスを使うこと
3. 自分のノートPCがちょうどすっぽり入るサイズのカバン・バッグ・リュックを約1万の候補から探し出す「HileSearch」http://gigazine.net/news/20150717-hile-search/
4. 本稿は執筆者（@yuzutas0）のブログ記事『PCが入るバッグを検索するWEBサービスを作りました』を大幅に加筆・修正して寄稿したものとなります。

# 第4章　「通知止まらんwww」体験アプリを作ったらバズって通知止まらんwww

　猫好きソフトウェアエンジニア、オクムラダイキ（@okumura_daiki）です！クリエイターの承認欲求を満たす「クソアプリ」を作ってみました。「クソアプリ」の楽しさと「クソアプリ」を作るときの意外な苦労をお伝えできればと思います。

## 4.1　はじめに

　SNSでよさげな投稿をしたのに、全然いいねされなかった、という経験はありませんか？「これは面白い」「たくさんいいねされるに違いない」とウキウキしながら投稿したのに、思ったより反応が貰えなかった……。

　悲しいことに、このような出来事は今日も世界中で起こり続け、毎日たくさんの人々を苦しめています。SNSで辛い思いをする人を1人でも減らしたい。私はそう思い立ち、アプリで世界を救うことを決意しました！

## 4.2　作ったもの

図4.1: 通知止まらんwww


　自分のツイートにたくさんの人が「いいね」したかのように、大量の通知が表示されるWebアプリ[1]です。GitHub[2]でコードを公開しています。

　渾身のツイートをしたけど誰からも反応がなかったときなどにご利用ください。バズった気分を簡単に味わうことができ、寂しさが紛れます。

　使いすぎには十分にご注意ください！電池を消耗するだけでなく、かえって虚しさが増すという事例も報告されています！（笑

## 4.3　苦労した点

iPhoneのUIをWEBサイトで再現するのに苦労しました。

ロック画面一つとっても「時計の文字だけフォントが細い」「上にスクロールした時に時計と日付だけスクロール量が違う」など、再現できなかったものも含めて、日常の利用では気付かない細かいところまで、本当にこだわって作られています。

iPhoneのロック画面に似せるために工夫した点を下記に紹介します。

文字フォント

CSSのfont-familyを下記のように指定することで、iOSのUIと同じフォントを使うことができます。

```
font-family: -apple-system, BlinkMacSystemFont;
```

すりガラス風の表現

通知カードの背景のすりガラス風の表現には、cssのbackdrop-filterプロパティを使用しています。

```
background-color: rgba(255, 255, 255, 0.5);
backdrop-filter: blur(12px);
-webkit-backdrop-filter: blur(12px);
```

通知のアニメーション

アニメーションは、classの追加によって CSS Transition が効くようにしています。ボワンとした動きは cubic-bezier を使って表現しています。

```
/* 初期値 */
.notification .notification-card {
  transition: transform 1s cubic-bezier(.36, .98, .31, 1.07);
  transform: scale(0)
}
/* class に active-enter を追加するとアニメーション開始 */
.notification.active-enter .notification-card {
```

```
    transform: scale(1)
  }
```

React.jsで実装したので、通知カードを追加した時のclassの追加にはreact-transition-group[3]というライブラリーを使っています。公式のドキュメント[4]に似た例があったので参考にしました。

### 全画面表示

htmlに下記のmetaタグを追加することで、ホーム画面から起動したときに全画面で表示されます。

```
<!-- ビューポートを端末の幅にして拡大縮小させない -->
<meta name="viewport" content="width=device-width, initial-scale=1,
 minimum-scale=1.0, maximum-scale=1.0, viewport-fit=cover">
<!-- URLバー等を非表示にする -->
<meta name="apple-mobile-web-app-capable" content="yes">
<!-- ステータスバーを透明にする -->
<meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
```

### 懐中電灯ボタン、カメラボタン

懐中電灯ボタンとカメラボタンは、3Dタッチに反応して、強く押すとアイコンが大きくなります。JavaScriptで touchforcechange というイベントを取得して3Dタッチを扱うことができます。

## 4.4　まさかの大反響

ただのネタアプリなのに、とんでもないことになってしまいました。

・「お前のツイート伸びすぎww」を完全再現　Twitterの通知が止まない状況を体験できるアプリが楽しい - ねとらぼ[5]

・「ちょw通知止まらんww」あなたもTwitterの“バズり現象”を体験できるアプリが楽しい！ - FNN（フジ・ニュース・ネットワーク） [6]

・ジャスティン・ビーバー気分？ 誰でも“バズる”経験ができるWebアプリが公開 - AbemaTV／『けやきヒルズ』[7]

・ツイートがバズった気分になれる、Twitterで通知が止まなくなった状態を再現するアプリ - INTERNET Watch[8]

livedoorニュース[9]やYahoo!ニュース[10]にも掲載されました。後者は本書執筆時点で掲載期間が終了しています。

開発者である自分のもとには多くのコメントが寄せられました。本当に通知止まらんwww

## 4.5　おわりに

　作ろうと思ったきっかけは、知人の「クソアプリアドベントカレンダー」という企画に参加したことでした。世の中に貢献しない「クソアプリ」を出し合って遊ぶ企画です。

　何か作りたいけどいいアイディアがない、という人にはクソアプリ開発がオススメです。「人に使いやすいように」とか「有用で便利なものを」といったことを気にせずに、好きなものを作れるので気楽です。今回の開発も、自分の使いたい技術を勉強しながら作りました。

　クソアプリを作ったのだからクソだと言われても傷つきません。むしろ褒め言葉として受け取れます。公開後は「くだらない」「草www」「いいクソアプリ」「こんなクソアプリ作りたい」などの感想をいただけました。作った甲斐がありました。

　今後もさらにくだらないものを作っていきたいです。[11]

```
オクムラダイキ / @okumura_daiki


猫好きソフトウェアエンジニア（ソフトウェアエンジニャア）です。
Advent Calendar に便乗して毎年クリスマス前にクソアプリを作っています。
大手SIer → DeNA → フリー。


https://qiita.com/okumura_daiki
```

1. https://okmr-d.github.io/fake-notification-web-app/
2. https://github.com/okmr-d/fake-notification-web-app
3. https://github.com/reactjs/react-transition-group
4. https://reactcommunity.org/react-transition-group/transition-group
5. https://nlab.itmedia.co.jp/nl/articles/1812/03/news103.html
6. https://www.fnn.jp/posts/00395960HDK
7. https://times.abema.tv/posts/5370055
8. https://internet.watch.impress.co.jp/docs/yajiuma/1156650.html
9. https://news.livedoor.com/article/detail/15692579/
10. https://headlines.yahoo.co.jp/hl?a=20181206-00010006-fnnprimev-life.view-000
11. 本稿は執筆者（@okumura_daiki）によるQiita記事『「通知止まらんwww」を体験できるアプリを作った』を加筆・修正して寄稿したものとなります。

# 第5章　積読の解消を促すWebサービスをリリースした話

どうも、フリーエンジニアの「きらぷか」(@kira_puka)です！「積読（つんどく）ハウマッチ」[1]を個人開発しました♪2019年4月に開発を始めて、2019年7月にリリース。AbemaTVにも取り上げていただき、2019年10月には登録者が1,000人を超えました。開発のきっかけやプロモーションの工夫をご紹介します。

## 5.1　サービス内容

　積んでる本の総額がわかる読書管理サービスです！積読が多い自分を戒めるためのWebサービスです(゜д゜)!

図5.1: サービス概要画面


　メインの画面はこんな感じです。本を登録して、ステータスを変更し、積読を管理するサービスになっています。

他の読書管理サービスと異なるポイントは「本の総額がわかる」ところ！いくら分の本を積んでいるかわかってしまいます。積ん読の恐ろしさを感じることができます。

## 5.2　きっかけとコンセプト

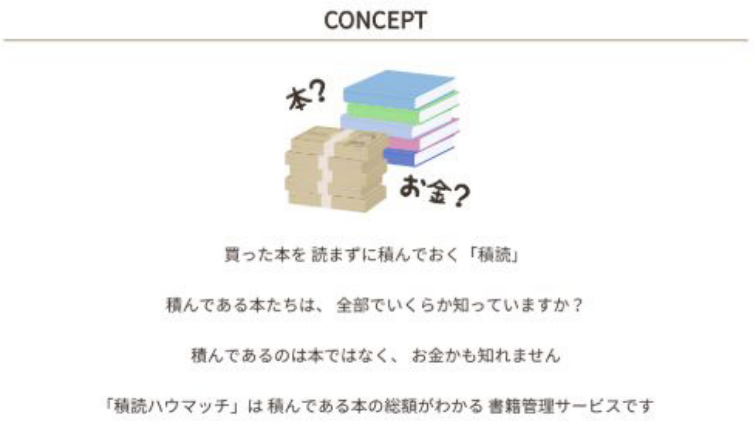
　サービスを思いついた時のツイートです。特にドラマチックな事件があったわけでもなく、完全にネタ的な思いつきで始まっています。

図5.2: Twitter画面


　作り始めは思いつきでしたが、もともと積読が多いことを気にしていました。

　サービスを思いついたのは、部屋に積み上がっている本を見た時です。ふと「これって全部でいくらなんだろう？」と思ったことがサービス開発の発端になっています。

図5.3: コンセプト画面



積読は、読まない限り、お金を積んでいるだけだと感じたのです。

## 5.3　使い方

使い方はシンプルです。
検索をして、

図5.4: 1画面


本を積んで、

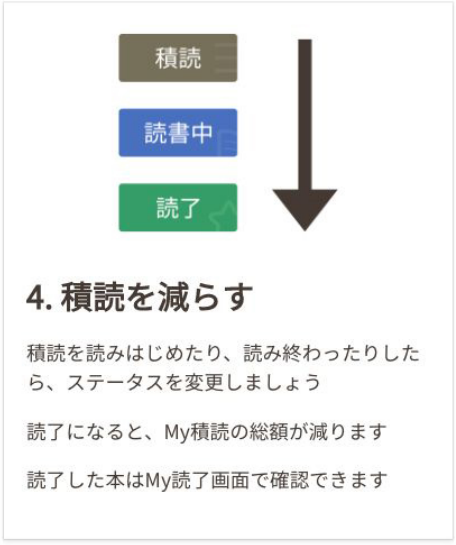
図5.5: 2画面

提示された総額を確認して、現実を受け止め、

図5.6: 3画面


がんばって積読を減らす！

図5.7: 4画面




持っている本を登録して、読み始めたり、読み終わったりしたら、ステータスを変更できます。

## 5.4　ランキング機能

積んでいる金額を見るだけだとつらいので……なんと、ランキング機能もつけました(´ω｀)

図5.8: ユーザーランキング画面


ランキング機能の誕生は「自分はこんなに積んでいるけど、他の人はどのくらいだろう？」と考えたことがきっかけです。他の人の積読を見て「まだまだ積んでも大丈夫！」と思いたくて……。

図5.9: 本ランキング画面


「読まれている本」「買われている本」のランキングもあるので、最近人気の本もわかります！

他のユーザーによって、新しく積まれたり読まれたりした本は、新着欄に表示しています！ぼーっと眺めるだけでもたのしい(´ω｀)

## 5.5　Twitterでシェアして積読金額を晒す機能

　SNSでシェアした際に表示される画像にも力を入れてみました。積んでいる金額を晒すことで、より読まないといけない焦りが生まれるかも？

図5.10: 積読おじさん画面


　（積んでいるのに、ドヤ顔のおじさんが登場します……）
　読んだ本の金額も共有できるので「こんなに読んだよ！がんばったよ！」もシェアできます。独学や自己研鑽をがんばっている人にもオススメです！

図5.11: 読んだおじさん画面

（こっちは本を読み終えてすっきりしたおじさん…）

## 5.6　リリースした後のプロモーション

リリースしてからはこんなことをしました！

・登録サイトに投稿・依頼

・いろんなところで記事を執筆

・進捗・使い方・Tipsに関する投稿

プロモーションをいろいろ試して、サービス利用者を増やしています。

## 5.7　登録サイトに投稿・依頼

「Webサービスを紹介するサイト」に登録しました！実際に使ったのはこの21サイトです！

| | サイト名 | URL |
|---|---|---|
| 01 | 開発会議 | https://devtalk.jp/ |
| 02 | Service Safari | http://www.service-safari.com/ |
| 03 | NewAppPlace | https://www.newappplace.com |
| 04 | SeekUps | https://seekups.seekgeeks.net/ |
| 05 | startapp | http://startapp.jp/ |
| 06 | Applishow | https://applishow.com/ |
| 07 | ツクログ | http://creators.eightbit.jp/ |
| 08 | Webサービス集めました | http://webatume.net/ |
| 09 | 放課後アプリ部 | http://houkago-no.appspot.com/ |
| 10 | Eggineer | https://www.eggineer.com/ |
| 11 | makepost | https://www.makepost.net/ |
| 12 | LITFIRE | https://litfire.jp/ |
| 13 | NewLaun.ch | https://newlaun-ch.com/ |
| 14 | ロケットリリース | https://rrws.info |
| 15 | Apps Times | https://appstimes.jp/ |
| 16 | creive【クリーブ】 | https://creive.me/ |
| 17 | APPLA | https://www.appp.la/ |
| 18 | WebFolio | http://websv.info/ |
| 19 | WebService Love!! | https://webservice.love/ |
| 20 | GOODINNO.press | https://goodinno.press/ |
| 21 | WEBサービスまとめてみた | http://webmatome.info/ |

大変なので、少しずつ行いました。

特に「開発会議」にはアップデートを投稿できる機能もあるので、更新したら登録しておくと公式アカウントでツイートしてもらえます！さらに、褒めてもらえる！(´ω｀)

図5.12: Twitter画面


各サイトで掲載されるので、それぞれからの流入や、検索でヒットすることを期待します。

## 5.8　いろんなところで記事を執筆

主に以下のところで関連記事を書いています。

・Qiita：技術記事[2]

・note：技術じゃない記事[3]

・Crieit：技術記事・個人開発記事・クロス投稿[4]

積読ハウマッチのユーザーは、いまのところエンジニアが多く、Qiitaの記事からの流入が多いです。

### Qiita

技術的な記事を書くためのサイトなので、開発で困ったことや得た知見をまとめています。実例としてWebサービスのリンクを張っておくと、そこから見てもらえるかもしれません。

Qiitaのトレンドに乗るとTwitterボットが拡散してくれることも！タグで記事を収集しているボットもいるので、ちゃんとタグを付けると拡散されやすいようです。

### note

noteの利用者にはエンジニア以外の人も多くいます。新着記事やタグから、エンジニア以外の愛読家の流入を期待しています。

Qiitaに比べて拡散されにくいものの、違う業種の人へは届きやすいようです。AbemaTVで紹介されたのも、noteを見ていただいたのがきっかけだとか。

### Crieit

プログラマー、クリエイターが何でも気軽に書けるコミュニティです。Qiitaよりもラフに色々書けるので、とても書きやすいです。ポエムっぽいのでも問題ないと思います。

個人開発のユーザが多いため、拡散されやすい傾向もあるように感じます。リリース記事を書いたら、いろんな人が拡散してくれました(´ω｀)

他にはない利点として、クロス投稿の仕組みが挙げられます。別のサイトで書いた記事も投稿できます。はてなブログやQiitaの記事をCrieitにも投稿できるので、併用して記事を書くように心掛けています！

## 5.9　進捗・使い方・Tipsに関する投稿

基本的にTwitterを利用して投稿を行なっています。

進捗や悩みをつぶやく

「これからこんな機能がでますよ！」とお知らせ投稿することによって、リリース前のコメントが利用者へ届くことを期待しています。

いまはサービスに興味がない人にも「リリースされたら使ってみようかな？」「この機能ができたら使ってみようかな？」と思って貰えたらいいなと考えて、進捗を投稿するようにしています。

また、リリースする前に良い・悪いなどの反応がわかるので、もらった意見を反映できるかもしれないなと考えています。

プロモーションに限らず、開発を継続している様子が伝わること自体も大事です。OSSライブラリーでも他のWebサービスでも、開発が活発なほうが、利用しやすいように感じます。

使い方やTipsをつぶやく

使い方やTipsなども動画つきで投稿しています。説明が不足していたり、ぱっと見わかりにくい部分も多いので、その点を補足する意味合いで使い方の説明を行います。

また、こんな使い方もあるのでは？という思いつきに関する投稿も試みています。サイト名だけでは積読がある人専用のサービスに見えてしまうので、ログインや積読がなくても楽しめるという点を発信しています。

## 5.10　おわりに

　いろいろ試していることをまとめてみました。これから何かを作ろう、リリースしようとしている人の一助になればと思います。

　個人開発ではやることがいっぱいです( ゜д゜)!!

　きっとまだまだ色々なことが足りていないと思うので、リリースを継続しつつ、プロモーションの手を次々と打っていきたいと考えています。

　眺めているだけでも楽しいサービスになっているので、まずはちらっと覗いてらえたらうれしいです♪面白そうと感じたら、ぜひ一緒に積み比べしましょう！良い積読ライフを！(´ω｀)[5]


きらぷか / @kira_puka

めもらばの開発担当兼代表。SIerからフリーエンジニアに転身。

受託開発をしながら、アプリ・Webサービス・ゲームなどいろいろ個人開発。

くらげのようにふわふわと、いろんなものを作ってます。

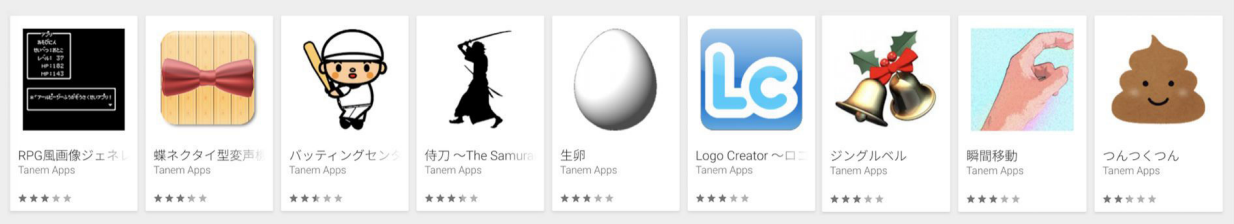
https://memory-lovers.com


1. https://tsundoku.site/
2. https://qiita.com/kira_puka
3. https://note.mu/kira_puka
4. https://crieit.net/users/kira_puka
5. 本稿は執筆者（@kira_puka）のブログ記事『【祝】積読を解消をうながすWebサービス「積読ハウマッチ」をリリースしました！』『個人開発したWebサービスをリリースした後にやったこと/やり続けていること』を加筆・修正して寄稿したものとなります。

# 第6章　累積120万DL！なぜかインドで大ヒット！クソアプリを量産しよう！

　クソアプリ開発者のTanemAppsです。8個のAndroidアプリを紹介します。すぐに真似できるお手軽ネタサービスだらけなので、ぜひアイデアの参考にしてください。

図6.1: https://play.google.com/store/apps/developer?id=Tanem+Apps



## 6.1　真実はいつも1つ！「蝶ネクタイ型変声機」

図6.2: 蝶ネクタイ型変声機


　使い方は簡単。声をアプリに吹き込んでみましょう。不思議なことに声の高さが変わります（機能はこれだけ）。もし怪しげな組織に薬を飲まされて、見た目が変わってしまっても大丈夫。きっとこのアプリが役に立つはずです。

　つい扇風機に向かって「我々は宇宙人だ」と言ってしまう。人間とはそういう生き物です。自分で声を吹き込む。いつもと違う音が耳に返ってくる。そんな独特のユーザー体験を、お手軽にアプリで実現できるのです。地味だけど、意外とハマりますよ！

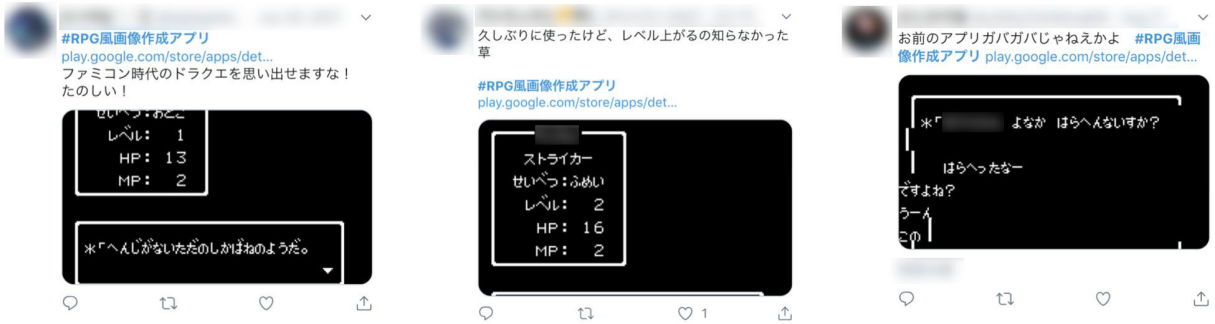
## 6.2　ほろびこそわがよろこび！「RPG風画像ジェネレーター」

図6.3: RPG風画像ジェネレーター


　某ゲーム画面を簡単に再現できます。テキスト表示は有名な無料フォントを使いました。俗にドラクエ風フォントと呼ばれます。ゲーム好きなら一度は使う価値アリです。

　ちなみに、最強の仕掛け「シェア機能」で、作った画像はTwitter投稿できます。

図6.4: タイムラインを埋め尽くす #RPG風画像作成アプリ のハッシュタグ




　画像をシェアして、やりとりを楽しんでもらう。俺はそれをエゴサしてニヤニヤする！

　ツイートにはインストールURLを載せています。このURLが導線になるのです。ユーザーは作った画像を共有する。そのツイートを見たフォロワーも使い始める。みんなで一緒にワイワイする。友達とドラクエで盛り上がる

のと同じだ！

## 6.3　おらワクワクすっぞ！「瞬間移動」

図6.5: 瞬間移動


　このアプリは常識破りのアイデアでアンドロイダーにも掲載されました。[1]

図6.6: レビューサイト「アンドロイダー」に掲載


　使い方を説明します。スマホを持つ手が重要です。スマホを握ったまま、人差し指と中指を立てましょう。チョキではなく、しっぺ。人差し指と中指をくっつける。中学生男子がよくやる「デュクシ」の構え。そのまま指先を眉間につける。これが瞬間移動のポーズです。指先に「気」を込めるのです。

　ここでアプリの出番です。念じながらアプリを使ってください。「ピュン」という音がなります（機能はこれだけ）。俺！今！瞬間移動してる！そんな気分になります！

## 6.4　野球好きなあなたに！「バッティングセンター ～無限素振り～」

図6.7: バッティングセンター ～無限素振り～


あなたが野球好きなら分かるでしょう。もしくは学生時代の野球部員を思い出してください。何もないところでバットを振るマネをしていたはずです。俺はよくやっています。

そこで思いついたわけです。何も持たずに素振りの猿真似。それだけじゃ味気ない。バットでボールをかっ飛ばす、あの快感。スマホなら再現できるのではないか。

ユーザーがスマホをバットのように振る。リアルな手応え（バイブレーション）。キーンという音も鳴る。いわばバッティングセンター！狭い家でも味わえる爽快感！

圧倒的閃き・・・！無限素振りの悪魔的奇手っ・・・！思いついたら・・・開発だろ・・・！開発するのがクリエイターってもんだろっ・・・！

## 6.5　サムライが好きなあなたに！「侍刀 ～The Samurai Sword～」

【遊び方】

端末を持って刀のように振る！それだけ！

画面に指を触れたまま左右に動かす（スワイプ）すると、刀の種類や音を変更できます。

音量やバイブレーションの有無は、設定メニューから変更できます。

・・・たくさん振ってるうちに、あの刀も使えるようになるかも？！

#タグ：侍,サムライ,さむらい,刀,カタナ,かたな,武士,剣道,リフレッシュ,ストレス解消

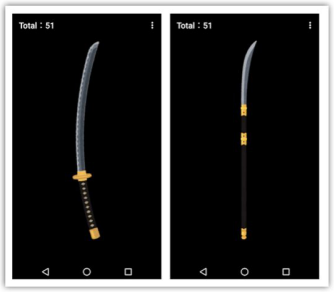
#TAG：samurai,katana,bushi,kendo,refresh

図6.8: 侍刀 ～The Samurai Sword～

侍刀 ～The Samurai Sword～

Tanem Apps　エンタメ

3+




　刀のコレクション要素など、射幸心を煽る仕掛けが満載です。システムの作りはバッティングセンターと同じで、振り方（センサー）と音と画像を差し替えただけ。

　一度アプリを作ると類似アプリは簡単に量産できます。例えば、ボクシングやゴルフのアプリならすぐ作れるでしょう。中学生男子は、蛍光灯の紐でボクシングをする生き物です。あれは親に怒られます。でもアプリなら許される。健康アプリとか適当に言い訳すればいいのです。体を動かして遊ぶ。いわばスマホ版「Wii U」です。

## 6.6　鳴らして遊ぼう！「ジングルベル」

図6.9: ジングルベル

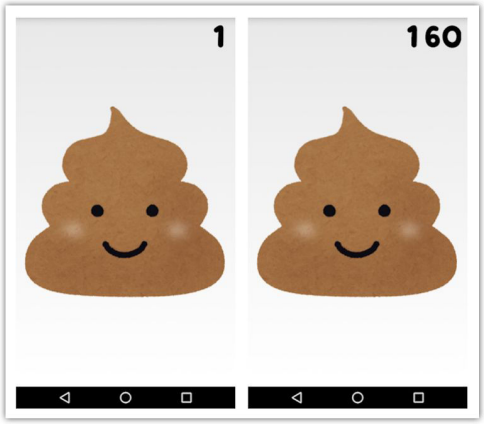

　ただベルの音がなるだけのアプリです。騙されたと思って小さいお子さんに渡してください。いつまでも楽しんで鳴らし続けるはずです。

　楽器やおもちゃを買うのは大変です。お金もかかる。場所も取る。だからこそ、こういうアプリが活躍するのです。考えてみてください。音を鳴らすだけのアプリなら、自宅から一歩も出ずに、ちょっとした時間で、お金も掛けず、簡単に量産できます。

　何よりも、自分が作ったアプリを、自分の子供が楽しく遊んでくれる。個人開発者にとって最高の時間です。他のユーザーなんて関係ない。市場規模やニーズなんて知らない。子育てエンジニアとして、胸を張って、このクソアプリを紹介しますよ！

## 6.7　正真正銘のクソアプリ！「つんつくつん」

図6.10: つんつくつん


つんつんすると、うんちが揺れる。それだけのアプリです。子供はうんちが大好きです。尋常じゃなく喜びます。このアプリを見ると、子供は確実にうんちをつんつんします。すると画面のうんちが揺れます。押せば揺れる。タッチという直感的なインターフェース。揺れるという感動的なエクスペリエンス。子供は満面の笑みになります。

さて、当初の計画を告白しましょう。「うんちのコレクション要素」です。いろんなキャラを登場させる。子供はキャラクターが好きです。しかも、うんちのキャラクター。心を鷲掴みにするはずです。ゆくゆくはシリーズ化。LINEスタンプに展開。やがて映画化。子供が欲しがれば、親の財布は緩むはず。

だけど俺はやりませんでした。なぜだか分かりますか。飽きちゃうからです。クソアプリを量産したい。うんちアプリだけに構っている暇はないのです！

## 6.8　埼玉の食卓から、世界のShokutakuへ

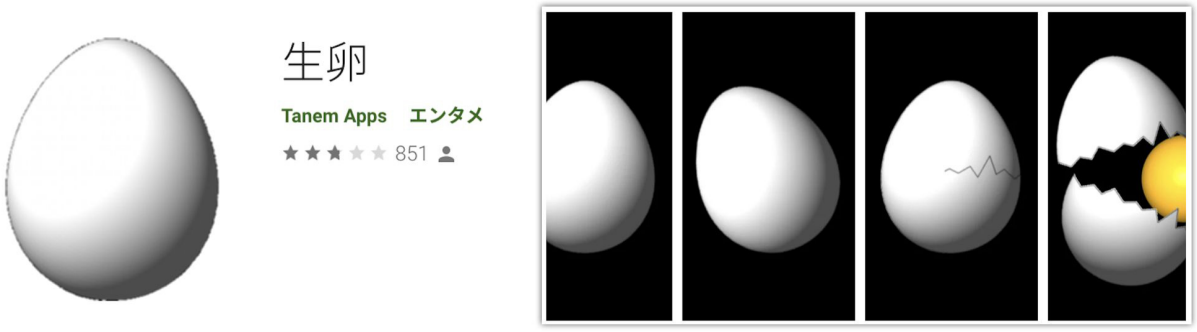
図6.11: TanemAppsのインストール数

| アプリ名 | 累計ダウンロード数 | 現在のユーザー数 |
|---|---|---|
| 生卵 | 42万 DL | 5000人 |
| Logo Creator | 40万 DL | 1万7000人 |
| 蝶ネクタイ型変声機 | 23万 DL | 5000人 |
| 瞬間移動 | 15万 DL | 3400人 |
| RPG風画像ジェネレータ | 2500 DL | 650人 |
| ジングルベル | 1700 DL | 80人 |
| せやな | 240 DL | 12人 |
| スリープボタン | 180 DL | 40人 |

さて、人気トップの「生卵」をご紹介します。

「生卵」の説明文だァーッ！！！

図6.12: 生卵




卵が好きすぎて作ってしまいました。

生卵を無限に割り続けたいすべての人々へ捧げます。

たまに、卵の中から黄身じゃない別の何かが出てくるかも。

```
Tag: Egg , 卵
```

「生卵」誕生秘話だァーッ！！！

　なにか作れないかな。スマホを持ちながらアイデアを考えました。片手で振り回せるアプリがいい。野球のアプリも、侍のアプリも、こうして作りました。今度は「スマホで卵を割ったら面白そうだ」と閃いてしまったのです。自分の才能が怖い。

　俺は卵かけご飯が好きだ。卵が好きだ。卵への愛が深すぎて、もはや卵の有無は関係なく「卵を割る」という行為まで愛してしまったのです。

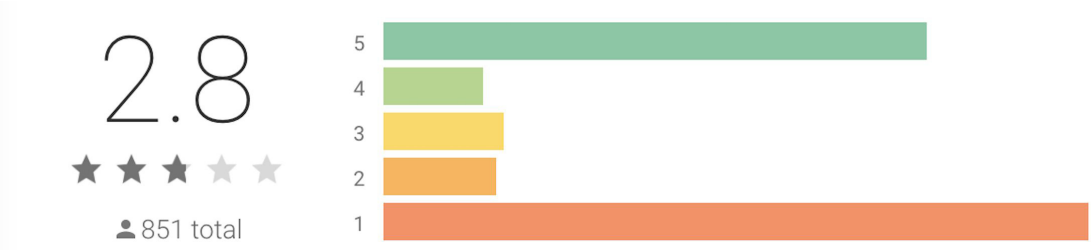
　無限にプチプチするだけのアプリがむかし流行りました。あれを卵でやろう。ひたすら割り続けられるじゃん。こうして卵を割るだけのアプリを作ることになりました。新しいアイデアとは、既存のアイデアの組み合わせなのです。

「生卵」ヒットの理由！？謎だァーッ！！！

　リリース1ヶ月後、「生卵」のインストールが急増。インドで大ヒットしたのです。毎日1万インストール！そして7000人がアンインストール！クソアプリですからね。これが何ヶ月か続きました。数十万円儲かりました。

　ヒットの理由が気になるでしょう。インドの有名人が紹介してくれた、というのが俺の予想です。ごめんなさい。あくまでも予想です。検索したけど、インドの言葉、読めません。どのサイトを探せばいいのか分かりません。なぜヒットしたのかは謎のまま……。

図6.13: 生卵のレビュー概要

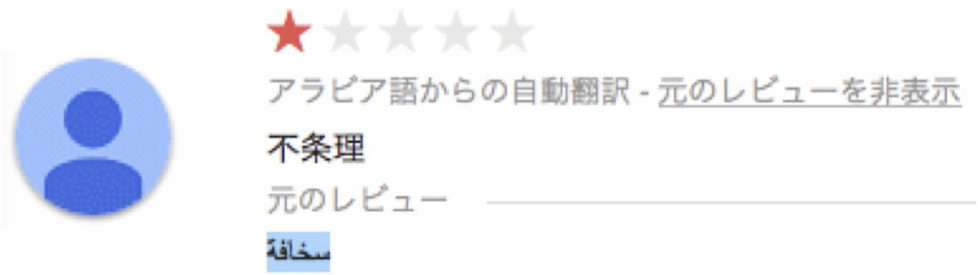



　アプリレビューは山ほど寄せられました。評価は基本的に「1」か「5」です。まぁ、いつものことです。俺が作るのはクソアプリですからね。

「生卵」ユーザーの感想！？カオスだァーッ！！！

問題はレビューです。読めない。アラビア語。翻訳したら「不条理」と表示されました。要望をちゃんと書いてほしいなぁ！このレビューで星1のほうが不条理だよ！

図6.14: 生卵のレビュー例




他のレビューも紹介します。海外ユーザーから「黄身の色、もっとオレンジっぽくしてくれ」と厳しいご指摘。お、おう。よし分かった。いいだろう。OKだ。こんな黄身じゃグローバルには通用しないよな。すぐに黄身のカラーコードを修正。クオリティが跳ね上がりました。100点満点中1点だったものが3点くらいになったかな。3倍だ。すごい。

「生卵」を産んだのは俺だァーッ！！！

レビューで1がつくのは別にいいのです。問題はそこじゃない。後発が出ました。卵を割るアプリ。パクリ。よくある話ですが、そっちがバズってしまいました。

そのアプリの勢いはすごかったです。目覚ましテレビが取り上げました。卵かけご飯を食べながら、俺はその放送を見ていました。アナウンサーの丁寧な解説は、アラビア語の短文レビューより分かりやすかった。俺が先だよこのやろう！

そんな絶望の中で、捨てる神あれば拾う神あり。あるブログがこっちのアプリを紹介してくれたのです。ブロガーさんはすごい。徹底リサーチで、こっちがオリジナルだと気付いたのでしょう。だからこっちを紹介してくれたのでしょう。分かってるやつは分かってる。世の中捨てたもんじゃない。そんなことを考えました。

しかし、現実は残酷でした。すぐにその記事が消されたのです。ちょっと！ブロガーさん！こっちが先だよ！間違ってないよ！記事を消さなくていいんだよ！

## 6.9　あくまでクソアプリ

まぁ俺もパクりだらけ。お互い様です。あくまでクソアプリ。120万DL - 世界を魅せるコンセプト。99%アンインストール - 世界が叩くコンテンツ。クソアプリ万歳！

クソアプリを支える技術

主にAndroidです。iPhoneアプリは作りません。Appleの登録料でお金を取られます。無料で作れるなら無料で作ったほうがいい。WEBサイトも作りません。アプリが好きです。スマホを持って手を動かす。指でつんつんする。そういう使い方が楽しいです。

いちおう「生卵」だけは例外でSafari（ブラウザ）版があります。iPhoneユーザーから要望が多かったからです。調べてみるとSafariは加速度センサーが使えるらしいので、せっかくならと作りました。Google Apps Engine が出た当初に、無料枠で運営しました。

メンテナンスはせずに完全放置です。どのアプリも、そもそもメンテ不要です。画像を表示するか、音を出すだけ。OSのバージョンアップ後も動き続けます。

アプリの枠を超えて - 最新のクソサービス「おにぎり君」

最近はLINEボット「おにぎり君」を作りました。流行に乗ってAIを使っています。

AIを自分で組んだのかって？そんな難しそうなこと、俺にできるわけないですよ！まじめに勉強したら楽しく作る時間がなくなります！クソアプリにあるまじき努力！勉強せずに、頑張らずに、お手軽に、ノリで、楽しく、ささっと作るのです！

というわけで、チャットボットは公開APIを使っています。なんかすごい研究所を持っている、なんかすごい企業で、なんかすごい人たちが、なんかすごいAPIを作ったらしい。それを使わせてもらうわけです。なんかすごい賢いAIなんだろな！

いちおうイラストは自分で描きました。見てくれこのクオリティを！

図6.15: LINEボット おにぎり君


右がテストユーザーのコメントで、左がLINEボット「おにぎり君」です。全然会話なりたたねぇなこいつ！(笑)

まだ、ここにない、くだらないアプリ

モチベーションは何か。世界中の人に使ってもらいたい？一発当てたい？俺は違います。需要は考えません。そうじゃない。これは個人開発です。俺が楽しいからやるんだ。

だから基本的には一発ネタ。飲みながら「こんなの作ったよ！」って見せる。笑いながら「アホじゃん！」って言われる。この段取りをやりたいだけです。

最後にTanemAppsの基本コンセプトを掲げます。

> **まだ、ここにない、くだらないアプリ（で世界中の人を笑かす）**

……というのは後付けです。ノリで始めただけです。そんなわけで、タメになる話もオチもないですが、これでおしまい。個人開発を楽しんでいきましょう！[2]

```
Tanem Apps

「まだ、ここにない、くだらないアプリ」を届けるAndroidエンジニア。自他共に認める
クソアプリ開発者。全世界で累計120万インストールを突破したが、あまりのクソアプリ
ゆえに99%のユーザーにアンインストールされている。でもそんなの関係ねぇ！

https://play.google.com/store/apps/developer?id=Tanem+Apps
```

1. 月間100万UUを超える人気を誇ったレビューサイト。GooglePlayストアのレビュー機能改善に伴い、2017年6月に「役割を終えた」として閉鎖。

2. 本稿はTanemApps（監修）との居酒屋談義を@yuzutas0が代理執筆したものとなります。

# 第7章　ダウンロード数と収入報告！スマホアプリの個人開発は儲かるのか？

　会社員Shiraです。50万ダウンロード突破の「CSV Viewer」をはじめとして、プライベートでアプリを作ってきました。広告収入やダウンロード数など、個人開発の実績を赤裸々に公開します。これからアプリ開発を始めたい！という人の参考になればと思います。

図7.1: アイコンがカワイイ自作アプリ


こちらは最も気に入っている自作アプリ「パッケージマネージャー」です。

Androidの個人開発をはじめた初期の頃にリリースしました。

アイコン画像は、地元の親友に頼んで描き下ろしてもらったイラストをもとに、私がGIMPという画像編集アプリで作成したものです。

## 7.1　スマホアプリの個人開発は儲かるのか？

　私は、システム開発の会社に勤めるサラリーマンでありながら、仕事で開発しているシステムに「そこまでの価値がないのではないか？」と疑問を持ってしまいます。

　脱サラしてアプリ開発だけで自由に食べていきたい。実際に稼げるか分からなくて不安だ。副業としてアプリ開発をやってみたいけど、時給に換算したらどうなんだろう？

　『アプリ開発』『個人開発』『広告収入』といったキーワードで、何百回も検索してきました。過去に情報を開示してくれた方々への恩返しも兼ねて、今回は私自身が広告収入やダウンロード数を公開します。

## 7.2　アプリ開発でのお金の稼ぎ方

実績報告の前に、アプリ開発でお金を稼ぐ方法を説明します。主に「有料アプリ」「アプリ内課金」「広告収入」の3種類で、組み合わせる事も可能です。

有料アプリの場合：広告はつけずアプリ内課金もなし。最初にダウンロードする時に一括でお金を支払い、その後はお金が不要なものが一般的です。

無料アプリの場合：広告を表示し、アプリ内課金にも対応しているものが多いです。

それぞれの稼ぎ方について詳しく説明します。

### 有料アプリで稼ぐ

Android向けアプリマーケット「Google Play」や、iPhone向けアプリマーケット「App Store」に有料でアプリをリリースし、それを購入（ダウンロード）してもらうことで利益を得る方法です。

Google PlayとApp Store、どちらも有料アプリを販売した場合には、取引手数料として売り上げの30%が徴収され、残りの70%が開発者へ支払われます。

例えば、100円のアプリが100ダウンロードされた場合には、100 × 100 × 0.7 = 7,000円が開発者に支払われることになります。1000ダウンロードなら7万円、1万ダウンロードなら70万、100万ダウンロードなら7,000万の収入を得ます。7,000万円も稼げるなんて夢がありますよね。

しかし、実際には100万ダウンロードなんて、個人開発者が狙うにはとてつもない数字です。

かの有名なファミコンの『ドラゴンクエスト』は、150万の販売本数だったそうです。100万ダウンロードが途方もない数字だということは、理解していただけると思います。

また、アプリの金額は開発者が自由に決めることができます。

マーケットには、2万円のアプリも存在します。2万円なら、ひとつダウンロードされるだけでその70%の14,000円を手に入れることができます。もしそんなアプリをリリースできるのであれば、とても稼げそうです。

### アプリ内課金で稼ぐ

アプリ内課金とは、プレイ中にお金を支払うことによって、レアなアイテムを手に入れたり、スタミナを回復するパターンのものです。大ブームを起こした『パズドラ』や『モンスターストライク』などが、このアプリ内課金に該当します。

スマホアプリでは基本的に課金しない派（無課金）の私も、アプリ開発の調査の一環として試してみたところ、パズドラでガチャに大分浪費しました。あの金玉が出て割れる瞬間のドキドキ感は最高ですね。

アプリ内課金はフリーミアム（Freemium）というビジネスモデルです。基本的には無料でアプリを提供し、機能の拡張や、より楽しむためのアイテムに課金する仕組みです。個人的にはこのフリーミアム戦略が一番稼げる方法だと思っています。

広告収入で稼ぐ

アプリの画面内に表示した広告をクリックしてもらうことで、収入を得る方法です。

以下のように、広告を配信する会社（アドネットワーク事業者）を利用します。

| アドネットワーク名 | 運営会社 |
|---|---|
| Admob | Google |
| MoPub | Twitter |
| Nend | ファンコミュニケーションズ |
| Amoad | サイバーエージェント |

ちなみに、私はAdmobしか利用したことがありません。運営会社はGoogleなので、海外展開しているAndroidアプリならば、Admobが最も安心だと思います。

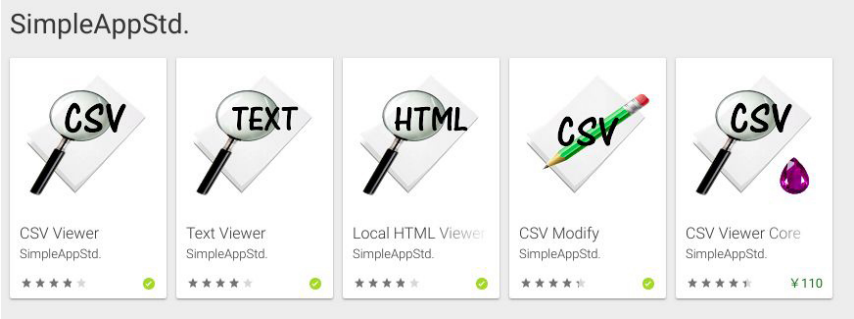
広告の種類もいくつかあります。

| 広告の種類 | 概要 |
|---|---|
| バナー | 画面の上部や下部に表示される長方形の広告 |
| インタースティシャル | ゲームクリア後などに表示される全画面広告 |
| 動画広告 | 広告の動画を見てもらうことで収益を得る広告 |

## 7.3　アプリのダウンロード数と収入報告

私は、Androidのスマホアプリを計6本リリースしています。

図7.2: SimpleAppStd. Apps




・内5本は無料アプリで、収益はAdmobによる広告収入です。

・内1本は有料アプリで、広告は表示していません。

見てお分かりの通り、ベースアプリ（CSV Viewer）を横展開、コスパ重視で開発したものです。ただし、ユーザーの目的は明確に異なり、それぞれにそれなりの需要があります。

ダウンロード数

執筆（2018年10月27日）時点の内容で、総ダウンロード数は約109万です。

| アプリ名 | 価格 | 総インストール数 | 平均評価/合計数 |
|---|---|---|---|
| CSV Viewer | 無料 | 584,580 | 3.93/1,882 |
| Local HTML Viewer | 無料 | 283,610 | 3.93/905 |
| Text Viewer | 無料 | 159,580 | 3.96/560 |
| CSV Modify | 無料 | 62,230 | 4.31/443 |
| PackageManager | 無料 | 3,190 | 3.73/11 |
| CSV Viewer Core | 有料100円 | 473 | 4.31/13 |

1番ダウンロードされているアプリは、他アプリのベースとなった「CSV Viewer」です。

お陰様で、50万ダウンロードを突破しました。

図7.3: CSV Viewer


ダウンロード数が伸びた理由は、以下の2つであると考えています。

・GooglePlay 内を「csv」で検索するとトップの方に表示される

・評価数が多い

上記を満たすためにそれなりの対策は行っています。しかし、決定的な勝因は、まだAndroidアプリが少なかった時期にリリースしたことだと思います。今から新規アプリを作って、ここまでダウンロード数を伸ばせる自信はありません。

広告収入

以下の表は、1ヶ月で発生する収益の内訳です。合計金額は21,723円です。

| アプリ名 | 広告収益 | 全体の割合 |
|---|---|---|
| CSV Viewer | 14,007円 | 64% |
| Local HTML Viewer | 2,214円 | 10% |
| Text Viewer | 4,975円 | 23% |
| CSV Modify | 513円 | 2% |
| PackageManager | 14円 | 0% |

リリース当初は数百円ほどだったものが、安定して2万円ほど稼いでくれるようになりました。積極的なアプリの修正はしていないため、不労収入と言っても過言ではありません。最近は「GDPR対策としてリリース先の国を絞る」「プライバシーポリシーを修正する」といった、若干の作業を行なっています。

## 7.4　ダウンロード数の考察

　リリースしたアプリがこんなにダウンロードされるとは、正直予想していませんでした。ダウンロード数が稼げた理由を次のように分析しています。

海外展開は必須

　稼ぎたいならば、海外展開は必須になります。
　各アプリにおける、日本と海外のダウンロード比率は以下の通りです。

| アプリ名 | 全世界のインストール数 | 日本のインストール数 | 日本の割合 |
|---|---|---|---|
| CSV Viewer | 323,662 | 16,549 | 5.11% |
| Local HTML Viewer | 252,625 | 20,255 | 8.02% |
| Text Viewer | 103,145 | 12,098 | 11.73% |
| CSV Modify | 38,676 | 1,290 | 3.34% |

　「個人開発なのに海外展開とかスケールが大きすぎる！」と気が引けるかもしれませんが、要するにやることは多言語対応です。私がリリースしているツール系のアプリは、全て「日本語」と「英語」に対応させています。
　英語が得意でなくても、google翻訳を使えばそれなりの英語になります！コツは、日本語を英語に翻訳した後、さらに英語を日本語に翻訳すること。読める日本語かチェックし、読めれば問題ありません。意味不明なら、微調整して再度翻訳を繰り返しましょう。
　今後は中国語にも対応させた方が良いのかもしれません。しかし、中国語はPCでの文字入力の仕方すらよく分からないため、やや敷居が高いです。（汗

ブルーオーシャンだった？

　欲しいツールをGooglePlayで見つけられなかった私は、ツールを自作してリリースしました。そのため、そもそも競合アプリが存在しませんでした。ダウンロード数を稼げた最大の理由は、この自作にあったと考えています。
　ということは、現在においても欲しいツールがGooglePlayに存在しない、イコール、自作してリリースすれば稼げる。という図式が成り立つ可能性は高いはずです。
　あなたが欲しいと思うツールを開発し、リリースしてみて下さい。稼げる可能性は高いですよ！

## 7.5　有料アプリの必要性について

広告がない有料バージョンの要望があり、『CSV Viewer』から広告を抜いた『CSV Viewer Core』を100円でリリースしました。しかし、これはマネタイズとしては失敗でした。

バージョンアップの度に両方のリリース作業が必要となるものの、有料版はあまりダウンロードされません。おそらく、有料版を購入してくれる方はアプリの使用頻度も高いので、100円分の広告クリックは容易に稼げるでしょう。そのため、広告を外したものを有料版としてリリースするパターンだけは、個人的にお勧めできません。

マネタイズを優先してアプリを開発する場合、基本バージョンアップの方向性はユーザーの要望に合わせるのではなく、自分が信じるポリシーに従うべきです。

## 7.6　アプリ開発は転職に有利

　アプリ開発を行うことによるメリットは何か？あなたがIT業界に身を置いているならば、最先端の技術と知識が身に付き、役に立つことは間違いありません。

　また、会社の仕事だけを行なっていれば、技術者として視野の狭い人間となってしまいます。ひとつの会社だけで生涯を終える人は少ない時代になっています。

　転職時には、「アプリをリリースしてます」と言うだけでも有利になりますよ。

## 7.7　アプリ開発の壁

　アプリの収入でサラリーマン並みに稼げるか？正直、かなり困難です。いかに素晴らしい技術を持っていても、いかに画期的なアイデアがあったとしても、稼ぐことは難しいものです。

　その理由は、プロモーションにあります。いくら良いものを開発、リリースしたところで、ダウンロードされなければ何も始まりません。プロモーションがうまくできず、せっかくリリースしたのにユーザーが1桁というのは、よくある話です。

　まずは、副業から始めることをオススメします。

## 7.8　アプリ開発で稼ぐために

結論としては、マネタイズを考慮したアプリをリリースし、運が良ければ儲かります。

私のアプリは、日本ではあまりダウンロードされず、海外の利用者が圧倒的に多いです。クリック単価を考慮すると、日本人の割合が多ければ、もっと収入が見込めます。

また、広告収入をメインとする場合には、ユーザーのアプリ使用頻度が最も重要になります。少ないダウンロード数でも、日本人をターゲットとし、定期的に起動させ、そこそこ利用するアプリであれば十分勝負できると考えています。

ただし、サラリーマンの収入ほどの稼ぎをアプリ開発で得ることは、相当難しいです。

## 7.9　まとめ

図7.4: https://play.google.com/store/apps/developer?id=SimpleAppStd.


要点をまとめます。

・ツール系のアプリを6つリリース

・運良くトータルで109万ダウンロード

・6年かけての総収入額は1,081,643円

・時給に換算すると可哀そうな金額

・ツール系アプリの広告収入は安定

・ダウンロード数を稼ぐなら海外展開は必須

・自分が欲しいツールを作るべし

・広告なしバージョンの有料アプリはマネタイズにならない

・アプリ開発、リリースで転職が有利になる

・運が良くて小遣い程度の稼ぎと考えるべし

・稼ぎたいならアプリを沢山リリースするべし

・稼げないアプリは早めに見切りをつけるべし

・広告アプリで稼ぐなら全画面広告は必須

・全画面広告の表示頻度は控えめにすべし

これらは、会社員Shiraの経験と実績から導き出した答えです。当然のことながら、全ての方に当てはまる内容ではありません。参考程度にとどめて頂けると幸いです。

## 7.10　最後に

プロモーションを学ぶために、私は現在、ブログ運営に注力しています！本格的なメディアとしてブログを育てつつ、アプリ開発を同時に行うことは、残念ながら至難の業です。よって、アプリ開発は一時中断することにしました。

あなたも私と同様に「ふたつの事を平行してやるのは難しい」「ブログ運営ではなくアプリ開発をメインでやるんだ！」と考えるならば、まずはTwitterから始めてみるのはいかがでしょうか？アカウントの育て方によっては、ブログ以上のプロモーションを期待できるかもしれません。リリースで満足せず、是非多くの人に使ってもらえるように行動していきましょう。

以上、『ダウンロード数と収入報告！スマホアプリの個人開発は儲かるのか？』でした。

```
会社員Shira / @ShiraAndroid

個人開発で不労収入を夢見る40歳のサラリーマン。プライベートでAndroidアプリを
リリース、累計100万ダウンロードを達成。36歳で未経験だったWEB系の会社に転職、
最終的には自由気ままなサービスを開発、運営して暮らしたい。そんな願望を垂れ流す
ブログ https://sastd.com を気ままに運営中。

https://play.google.com/store/apps/developer?id=SimpleAppStd.
```

# 第8章　格安スクレイピングを支える技術

　はじめまして、Webエンジニアの morizyun（以降、筆者）です。ブログ「酒と泪とRubyとRailsと」[1]を運営しています。3年ほど個人開発している「価格比較サイト」にて試行錯誤を繰り返した、格安スクレイピングのTipsや便利なホスティングサービスについて紹介します。

## 8.1　はじめに

「価格比較サイト」では、ECサイトのデータをキュレーションしています。サイトのコンセプトは、新品や中古の一番安い商品を誰でも簡単に見つけられることです。

このサイトは、2019年1月末時点で月間20万PV、約11万件の販売情報を収集しています。これらの大量のデータを相手側のサイトに迷惑をかけないように、最低限の頻度でスクレイピングしています。

2018年11月は4,431円のコストで運営しました。内訳は以下の通りです。

・Webサーバー（4GBメモリ）1台：2,270円

・S3による画像配信：624円

・スクレイピングサーバー（2GB メモリ）3台：1,537円

2年間のサービス開発を通して、サーバに費用をかけられないイチ個人開発者として、格安でスクレイピングを行う技術を模索してきました。

## 8.2　このパートで学べること

　ここで紹介する技術を駆使することで、以下のような「他のサイトからスクレイピングをするようなサービス」を簡単に作ることができます。

・興味のあるジャンルのニュースやお店をまとめるサービス

・自分の住んでいる地域の天気が悪くなったときにメールで通知を出すサービス

　前半パートでは、スクレイピングを実施する上で考えるべきポイントを紹介します。後半パートでは、筆者が実際に使っている技術とそのサンプルコードを紹介します。

想定する読者

　スクレイピングを伴うサービスの開発を行いたい開発者。特にChromium（Chrome）を使ってリッチなサイトのスクレイピングをしたり、定期的に複数サイトのスクレイピングを並行して行いたい開発者向けの内容です。

書くこと、書かないこと

　今回はスクレイピングで知っておくと良い知識、技術を中心に紹介します。実装の細かい内容には触れません。詳しい実装の質問等があれば、気軽に筆者Twitter（@zyunnosuke）に連絡を頂ければ喜んで回答します。

## 8.3　スクレイピングで気をつけるべきこと

スクレイピングの技術を紹介する前に、スクレイピングサービスを作る上で気をつけるべきことについて、筆者の考えを紹介します。

### 事例紹介：岡崎市立中央図書館事件

スクレイピングを行うにあたって、押さえておくべき事柄の一つに「岡崎市立中央図書館事件」があります。この事件はWikipedia[2]には次のように書かれています。

> 岡崎市立中央図書館事件（おかざきしりつちゅうおうとしょかんじけん）は、2010年3月頃に岡崎市立図書館の蔵書検索システムにアクセス障害が発生し、利用者の一人が逮捕された事件である。利用者に攻撃の意図はなく、また、根本的な原因が図書館側のシステムの不具合にあったことから論議を呼んだ。逮捕された人物が取調べの後、Librahackというサイトを立ち上げて解説をしたことから、Librahack事件とも呼ばれる。

ある開発者が図書館のWebサイトの蔵書検索の使い勝手に満足できず、「1秒に1アクセス」を行うクローラーを作ってデータを収集し、自ら蔵書検索を行うサービスを作ろうとしたところ、蔵書検索システムが特殊な仕様だったため、一般ユーザーがWebサイト側から閲覧できない状態になってしまいました。クローラー自体は、GoogleやYahoo!等の検索エンジンでも使われている手法です。「1秒に1アクセス」も議論の余地はありますが、裁判所は「業務妨害の強い意図が認められない」と判断しており、違法ではないと考えられます。問題点は、Webサイトを閲覧困難な状態にしてしまったことです。

スクレイピングを行うと、時としてこのような悲しい結果を生むことがあります。相手側のサイトに負荷を掛けないように、十二分に気をつけるようにしましょう。

### 法律面

筆者は法律面での専門家ではないため、ここでは詳細な説明は省きます。以下の弁護士さんの記事を参考にすることがオススメです。

・【スクレイピングと法律】スクレイピングって法律的に何がOKで何がOUTなのかを弁護士が解説[3]

### できるだけAPIを使う

WebサイトのHTML構造は、A/Bテストや UI改善で頻繁に変化します。せっかく作ったプログラムもHTML構造が変わると、それに合わせて修正が必要になります。

一方で、APIはリクエストとレスポンスの構造が安定しているので、プログラムでデータを取得することに適しています。APIを提供しているサービスでは、スクレイピングではなくAPIの利用がオススメです。

特に大手サイトは多くのAPIを開放しているので、ぜひ活用してみて下さい。

・Yahoo! Japan デベロッパーネットワーク[4]

・Rakuten Developers[5]

・2018 個人でも使える！おすすめAPI一覧[6]

## スクレイピング先とWIN-WINになる

スクレイピングをする「だけ」では、スクレイピング先にデメリットを発生させてしまいます。

・リクエストに対してHTMLを生成・返却するのでサーバーやDBに負荷がかかる

・リクエスト元がプログラムなので、アクセスが収益につながりにくい

一方で、Googleの検索システムもスクレイピングを繰り返すことで、検索用のデータベースを構築して、検索サービスを提供しています。多くのサイトはSEO（Search Engine Optimization：検索エンジンへの最適化）を行い、Googleのスクレイピングを歓迎しています。これはGoogleの検索システムを通じてサイトにユーザーの流入が発生するためです。

つまり、スクレイピングを行う場合には、スクレイピング先のサービスに「メリット」を提供することが大切になります。スクレイピングを伴うサービスを企画・開発する際は、スクレイピングによって得られる「自分のメリット」だけでなく、スクレイピング先が得られる「相手のメリット」を心がけましょう。

## 8.4　スクレイピングを行うプログラムの基本構造

スクレイピングを行うプログラムの、基本的な構造について紹介します。

### スクレイピング処理の要素

スクレイピングのプログラミングには、大まかに次の3つの要素があります。

・Process.1 相手先のサーバからHTMLを取得する

・Process.2 取得したHTMLからデータを抽出する

・Process.3 使いやすいデータの形に加工してDBに格納する

プログラムを作る前に、この3つの要素をどのように実装するか検討しましょう。ちょっとしたデータの取得ならば、まとめてひとつのプログラムで処理しても構いません。

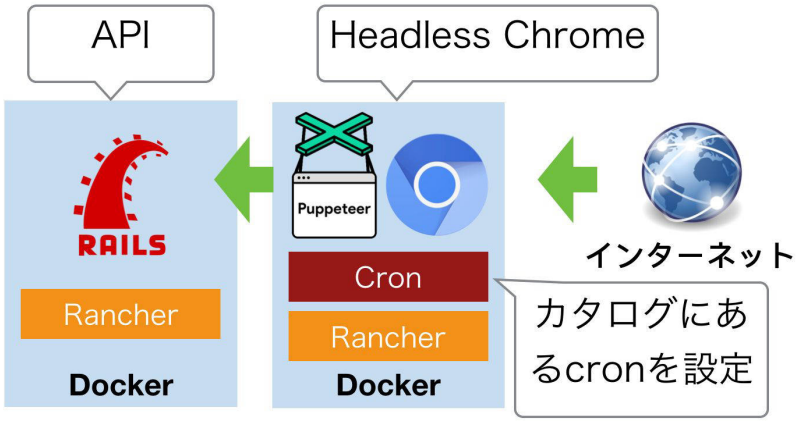
大規模なスクレイピングを行う場合は、それぞれの処理を分割したり、データを頻繁にDBやストレージに保存しましょう。プログラミングを分散・並列化できたり、データの再利用性を高めることができます。

### 事例紹介：筆者サイトのアーキテクチャ紹介

スクレイピング処理の事例として、冒頭で紹介した筆者サイトのアーキテクチャを紹介します。ECサイトから販売情報をスクレイピングして、キュレーションした結果をWebサイトに表示します。

下図が大まかなアーキテクチャです。

図8.1: サイトのアーキテクチャ



「スクレイピングを行うプログラム」（Puppeteer）と「整形処理をしてデータをDBに保存するプログラム」（Rails）の2つを分けて実装しています。このアーキテクチャによって、Headless Chomeを分散して立ち上げて個別のサイトごとにカスタマイズされたスクレイピングを行いつつ、筆者が使い慣れたRailsで複雑なビジネスロジックの実装を実現しています。

「スクレイピングを行うプログラム」は、Headless Chromeを簡単に使える「Puppeteer」がオススメです。Googleが提供しているライブラリーなので、安心感があります。

一方で、「共通の処理をしてデータをDBに保存するプログラム」は、開発者が得意な言語・フレームワークがオススメです。共通の処理や複雑なロジックをサーバーサイドで一箇所で実装することで、実装コストを抑えることができます。

## 8.5　スクレイピングが捗るライブラリーの紹介

スクレイピングに役立つ、次のツールをサンプルコードとともに紹介します。

・Chromiumでのスクレイピング：Node.js, TypeScript, Puppeteer, Cheerio

・Scrapy Cloudでのスクレイピング：Python, Scrapy

・Google Action Script(以下GAS)でのスクレイピング：GAS, TypeScript, Clasp

・Dockerベースのインフラを簡単に作る仕組み：Rancher

・Mac上で定期的なスクレイピングをする仕組み：Bitbar

Puppeteer [Node.js]

Google謹製のHeadless Chrome用のNode.jsのライブラリー[7]です。Puppeteerの利用用途としては、次のようなものがあります。

・スクリーンショットやPDFの生成

・SPAのクロールとpre-renderedコンテンツの生成（SSR）

・フォームの投稿やUIのテスト、キーボード入力の自動化

・最新Chromeを使っての最新のJavaScriptやブラウザ機能の自動テスト

サイトのスクリーンショットを作成したい場合は、次のように記述します。

```
const puppeteer = require("puppeteer");


(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto("https://example.com");
  await page.screenshot({ path: "example.png" });


  await browser.close();
})();
```

このように、簡単なコードでWebブラウザーChromeを操作できる点が魅力です。詳細な使い方を知りたいときは、公式ページが充実しているのでオススメです。

Scrapy [Python]

スクレイピングに最適化された、Pythonのフレームワーク[8]です。効率的に質の高いスクレイピングを実現できます。Scrapyで作ったプログラムは「Scrapy Cloud」[9]というクラウドサービスにコマンドひとつでアップロードできるので、実装からデプロイ、実運用までが非常にスムーズです。

https://blog.scrapinghub.com の記事を取得するサンプルコードを紹介します。

```
# -*- coding: utf-8 -*-
import scrapy


class BlogSpider(scrapy.Spider):
    name = 'blogspider'
    start_urls = ['https://blog.scrapinghub.com']


    def parse(self, response):
        for title in response.css('.post-header>h2'):
            yield {'title': title.css('a ::text').extract_first()}


            for next_page in response.css('div.prev-post > a'):
            yield response.follow(next_page, self.parse)
```

たったこれだけで、ブログのタイトルが取得できます。Scrapyはミドルウェアや各種設定も充実しています。作成したコードは shub ライブラリーを使うと、簡単にScrapy Cloudにデプロイできます。SPAのようにJavaScriptが多用されたサイト以外が対象であれば、Scrapyは少ない工数で効率的にスクレイピングを実現することが可能です。

Google Apps Script [Node.js]

GAS[10]はGoogleが提供するJavaScriptベースの言語で、Googleのサーバー上に実行環境があります。そのため、GASでも簡単にスクレイピングを行うことができます。clasp[11]は、GASをローカルで編集、サーバーにデプロイするOSSで、Googleが提供しています。

Googleをスクレイピングして、タイトルを取得するコードを紹介します。TypeScriptでコードを書くことができます。

```
function myFunction() {
  const response = UrlFetchApp.fetch("https://google.com");
```

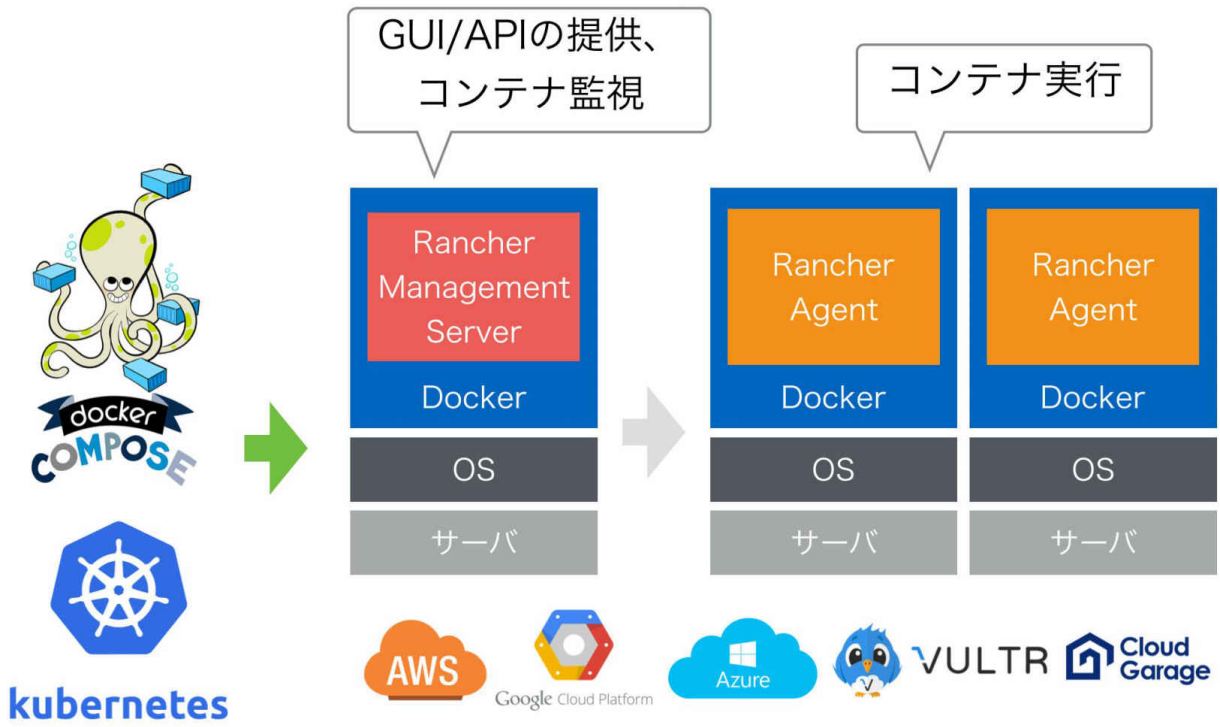
```
  const myRegexp = /<title>([\s\S]*?)<\/title>/i;
  const match = myRegexp.exec(response.getContentText());
  let title = match[1];

  title = title.replace(/(^\s+)|(\s+$)/g, "");
  Logger.log(title);
}
```

作成したコードは clasp コマンドで実行環境にデプロイして実行できます。Googleスプレッドシートとも連携できるので、集めたデータの編集作業にも使えます。

### Rancher

Kubenetesなどのツールをラップして、簡単にDockerをデプロイ・運用できるOSS[1][2]です。筆者はRancher + Docker経由でHeadless Chromeのプロセスを動かしています。

図8.2: Rancherの概要



Rancherの大まかな概要を紹介します。

・インフラはDockerが動けばどこでも動かせる(AWS, GCP, VPS, etc)

・Docker上でGUI/APIを提供する「Management Server」とコンテナ実行をサポートする「Agent」を動かす

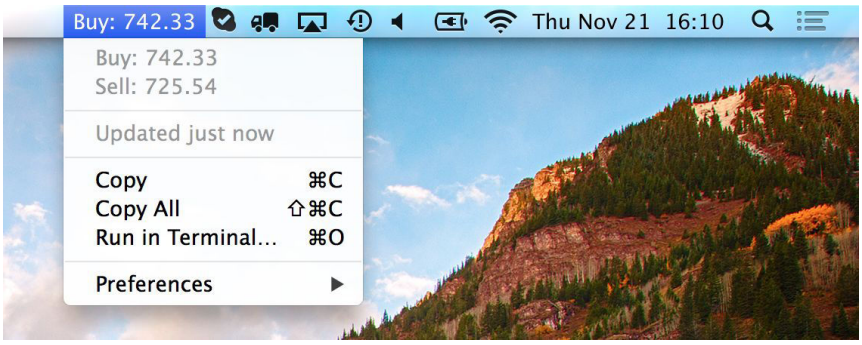
・ユーザーがDocker Compose等を投入するとコンテナがよしなに立ち上がる

・cronのPluginやWordPressなどの有名アプリが数クリックで公開できる

BitBar

Macのメニューバーへ簡単にメニューを追加できます。追加するメニューは、シェルスクリプトやRubyなど普段使い慣れた言語で記述できます。BitBarのスクリプトは定期的な実行をサポートしているので、普段使っているMac上でちょっとしたスクレイピングを定期実行したいときにオススメです。

図8.3: BitBarのプレビュー



## 8.6　おまけ：オススメのインフラ

　個人開発者が開発を行う上で、どこのサーバに何を使うかは重要な問題です。お財布を少しでも守るためにも、コストの安いサーバを選びたいものですよね。おまけとして私のオススメするサービスをいくつか紹介します。

Scaleway

　フランスとオランダのデータセンターで、格安のVPSやサーバーを提供してくれるサービス[13]です。2019年1月時点では、2GBメモリのサーバーを月額3.99ユーロ（約496円）で使うことができます。。1年ほど使っていますが、動作は安定しています。レイテンシーが気にならないのであれば、素晴らしい選択肢といえるでしょう

Google App Engine

　Googleが提供しているPaaS[14]で、アプリを簡単にデプロイできます。日本リージョンも提供されており、CDNも使うことができます。個人開発者が開発に使える時間は限られるので、インフラを気にせず、アプリの開発に集中できる点が魅力です。また、2019年1月時点では12か月間有効な$300のトライアル・クレジットがあり、お財布に優しい点も魅力です。

## 8.7　あとがき

筆者がスクレイピングの知見を得たのは、ひとえに「価格比較サイト」で試行錯誤を積み重ねたからです。リリースから2年間は売上ゼロでしたが、今では本業の収入を上回るまでに成長しました。ひたすら3年間、コツコツと土日にオフィスや自宅で作業をしています。

これだけ長く続けられたのは「安く買う」のが好きだからです。学生時代の筆者にはお金がありませんでした。ゲームやパソコンを1円でも安く購入できるように、ヤフオクで何日も探しては価格交渉をしました。努力の末に安く買えたときは、本当に嬉しかったです。趣味の延長で、安値・セール情報の紹介ブログを運営し、稼いだ収益を学費に充てました。

冷静に考えると、その時間でアルバイトをしたほうが、欲しいものを効率的に買えたかもしれません。しかし、当時の取り組みがあったからこそ、今の筆者の活動に繋がっています。価格比較サイトの運営も、格安スクレイピングの追求も「安く買う」という意味では同じです。

個人開発で収益を得られるようになると「企業で働く」「フリーランスになる」以外の新しい選択肢になると信じています。なにより、オーナーシップを持って自分のサービスの成長に全力を尽くすことができるのは、大きな成長のチャンスだと考えています。

Stay hungry, stay foolish with happy coding!

```
morizyun / @morizyune


技術ブログ「酒と涙とRubyとRailsと」を書いています。 https://morizyun.github.io/
現在はフロント・バックエンドエンジニアとしてフルタイムで仕事をしつつ、プライベート開発で
ECのキュレーションサービスを運営しています！個人開発が大好きです！
```


1. https://morizyun.github.io
2. https://ja.wikipedia.org/wiki/岡崎市立中央図書館事件
3. https://it-bengosi.com/blog/scraping/
4. https://developer.yahoo.co.jp/
5. https://webservice.rakuten.co.jp/
6. https://qiita.com/mikan3rd/items/ba4737023f08bb2ca161
7. https://pptr.dev/
8. https://scrapy.org/
9. https://scrapinghub.com/scrapy-cloud
10. https://developers.google.com/apps-script/
11. https://github.com/google/clasp

12. https://rancher.com/
13. https://www.scaleway.com/
14. https://cloud.google.com/appengine/

# 第9章　個人開発だけで食べられるようになるまでにやったこと

　プロ個人開発者のアタカ（@atagon）です。合計5つのiOS・Android向けアプリを運営しています。受託無し、個人開発だけで都心で暮らせる売上に到達したので、これまでにやったことをご紹介したいと思います。個人開発だけでの生活に挑戦したい！という方の参考になれば嬉しいです。

## 9.1　自己紹介

　ストアで「付箋」キーワード1位を取った「付箋todoメモ帳 QuickMemo+」[1]などのアプリを個人開発しています。QuickMemo+ では、本物の付箋メモ（ポストイット）のように、思いついたら簡単にすぐ付箋メモを貼り付けられます。

図9.1: QuickMemo+


　金融系のB2B開発を2年、B2Cのサービス開発を10数年経験した後、2018年3月に独立。会社員時代から個人でアプリを開発し、自由気ままに暮らせる生活を目指して奮闘してきました。コツコツとアプリの改善を続けて行った結果、2019年6月時点で、ついに個人開発だけで食べられるようになりました。

## 9.2　付箋アプリを作った理由

自分が好きなものをやる

　僕は付箋が大好きで、市場調査をするまでもなく、アプリを作るなら付箋アプリ！と決めていました。人生初の個人開発アプリを今も更新し続けていて、お陰でこれだけで食べていけるようになりました。

　市場調査をしても「自分にとって」ツマラナイ物になるだけかなぁと思っています。評価を受けられなくても、売上がなかなか増えなくても、自分の「好き」を軸に、「自分が欲しい物」を作って、「自分の為」に更新し続ける。好きで好きでずっと開発を続けられる物を作っていれば、いつか日の目を見るチャンスが来るはずです。

自分が出来ることをやる

　僕はアプリの開発となんちゃってデザインしか出来ません。だから、インフラ、サーバー、ウェブは一旦捨てて、自信のある技術の範囲内でサービスを設計しました。

　自分に出来ない事、自分がまだやっていない事は、魅力的に見えます。そこにこだわってしまうと、余計なコストや、諦める理由、時間の浪費に繋がります。

　いまいちアプリが伸びていなかった時、ブログ、VR、Unity、新アプリの企画と手を伸ばしてみました。だけど、それらは「挑戦」と言いつつ、ただの「逃げ」だったのかもしれません。明確な結果が出たのは、全てをやめて、既存アプリに集中してからでした。

1人で全部を回せるサービスを

　毎日コンテンツを提供して、人を雇って管理しないといけないようなサービスは避けて、「たまにお問い合わせに答えれば良い」ぐらいの状態を目指しました。どこにいても、遊んでいても、寝てても、お金が入る。そんな自動販売機のようなサービスです。

　個人開発は、企画、デザイン、開発、マーケティング、カスタマーサポート、全てを一人でやることになります。運用ができなかったり、ユーザー数が増えるほど赤字が膨らむようだと、長続きしません。付箋アプリなら上手く回るのではないかと考えました。

## 9.3　インストールしてもらうためにやったこと

分かりやすいアプリ名

図9.2: QuickTodo+


　アプリ名は「名前だけで機能が分かって」かつ「検索されそうな言葉」を入れています。TODOアプリでは「QuickTODO」[2]と必ず「TODO」と入れる。アラームアプリでは「しつこいお薬アラーム」[3]と必ず「アラーム」を入れる。良く分からないものは、検索もされないので、誰もたどり着けません。

ストア画像は横長一択

図9.3: 横長のストア画像


僕はこれでダウンロード数が40%伸びました。アプリが縦向きだろうと横向きだろうと、ストア画像は横長一択です。

2019年6月現在、iOSのAppStoreでは、ストア画像が縦長だと3つの画像が小さく表示されます。ところが、横長にすると、ひとつの画像だけが大きく表示されるので、インパクトは3倍です。ユーザーはひとつひとつの画像や文章を吟味しているわけではありません。画面を流し見しているので、インパクトを与えることが大事です。

アプリサイズは150MB以下に

iOSアプリでは、150MBを超えるとWiFi環境下でしかダウンロード出来なくなります。僕は一部の画像をpng8にしたりjpegの解像度を下げたり、インストール後にサーバーからダウンロードするなど、必ず150MB以下に収めるように工夫しています。

企業活動なら知名度や広告によってカバーできるかもしれませんが、個人開発では違います。キャリア通信でダウンロードできるかどうかは、とても大事なポイントです。

自分でレビューをつける

ストアレビューがない無名の状態では、あまりダウンロードされないように思います。そこで、アプリをリリースしたら、まず自分でレビューをつけました。また、奥さん、知人、友人みんなにもレビューをお願いしています。ただし、金銭など報酬を渡すのは規約違反なので、あくまで善意での利用とレビュー投稿を依頼します。

## レビュー依頼機能を付ける

ユーザーにとってレビューというのは面倒な作業です。ただ待っているだけでは、自然に増えたりはしません。なので、僕はメイン機能の実装が終わった後、レビューを促す機能を入れました。すぐ審査に出して1秒でも早くリリースしたい！という気持ちをグッとこらえながら。

## 星1レビューこそしっかりフォロー

自分の作ったアプリに星1のレビューが付くと丸1日凹みます。最初は、ただ心のざわめきが去るのを静かに静かに待っていたのですが、ある日、このショックから立ち直る効果的な方法を見つけました。

それは「問題を指摘してくれた事に感謝」して「しっかり返信」して、お問い合わせを促す事です。すると、結構な確率で星5に変えてくれました。

おそらくテレビに文句を言うのと同じノリだったのかもしれません。アプリの先には運営者がいるという事実に気付いてもらえるチャンスです。

## アプリの先に人を感じさせる

アプリの先に人を感じさせる事が、競争の激しいアプリ市場で生き残る鍵だと思っています。大手企業のアプリではテンプレートのような対応をされることが多いです。送った甲斐が無いと、二度と要望を送ることはなくなって、ユーザーの声が届かなくなります。逆に、毎回ちゃんと返信をすれば、積もり積もって大きな差別化にも繋がるはずです。

だから僕は、お問い合わせには即返信、レビューには全部違う返信、本名で名乗り、感謝の言葉を心がけています。バグ報告のメールが来たら「対応します」と返信するだけでなく、修正後に「対応しました」と伝えるところまでやります。要望を取り入れない場合も、曖昧にせず、取り入れない事をちゃんと伝えた上で「技術用語を使わずに」丁寧に説明しています。

丁寧に対応するだけで「ちゃんと返事が来ると思っていなかった！感動した！」という言葉が寄せられました。その効果はレビューに書かれるメッセージにはっきりと現れてきます。

## 長期運用も一つの差別化

採算が合わず、すぐ撤退する、競争の激しいアプリ業界。利用しているサービスの更新が止まる。ストアから消える。僕自身ユーザーとしてそんな体験を何度もしました。

だからこそ「続けていること」自体を価値としてアピールできます。僕のアプリもレビューに「数年使っています！」というレビューが増えてきました。その結果、口コミだけで毎日200以上ダウンロードされています。

## 9.4　使い続けてもらうためにやったこと

### 「使う理由」を生み出す

アプリを使い続けてもらうには理由が必要です。

・ゲームアプリを使う理由：魅力的なシナリオやアイテム。

・CGM（投稿）サイトを使う理由：ユーザーのコンテンツ。

・コミュニティーサービスを使う理由：ユーザー同士の交流。

これらはどれも、沢山の人（作り手・参加者）が必要です。個人で継続的に運営するのは厳しいと考えました。

僕は今のところユーティリティアプリだけを作っています。メモなどのアプリはユーザー個人で「使う理由」が完結しています。最小限の開発・運営で、毎日使ってもらえるアプリにできるはずです。

### プッシュ通知を活用

プッシュ通知を送ると、簡単に起動率を上げられます。僕は「アプリの機能解説」をプッシュで送っています。嫌がられて離脱されるような内容ではありません。むしろユーザーがアプリを使いこなせるようになります。無料で簡単に導入できるOneSignalというプッシュ配信サービスがオススメです。

### タップ数を極限まで減らす

数字に特化したヒアリングアプリ「Lisnum」[4]では、聞こえた外国語の数字を電卓のようなUIで入力してもらいます。



図9.4: Lisnum


リリース時は、間違った時に直すための「削除ボタン」と、答えを確定させる「回答」ボタンを用意していました。ところが、友人にテンポが悪いと指摘されてしまい「削除ボタン」と「回答」ボタンを無くすことに。正しい数字を入れた瞬間に正解判定を、間違った数字を入れた瞬間にNG判定を行うように変えました。結果としてスリリングでテンポの良いアプリになりました。

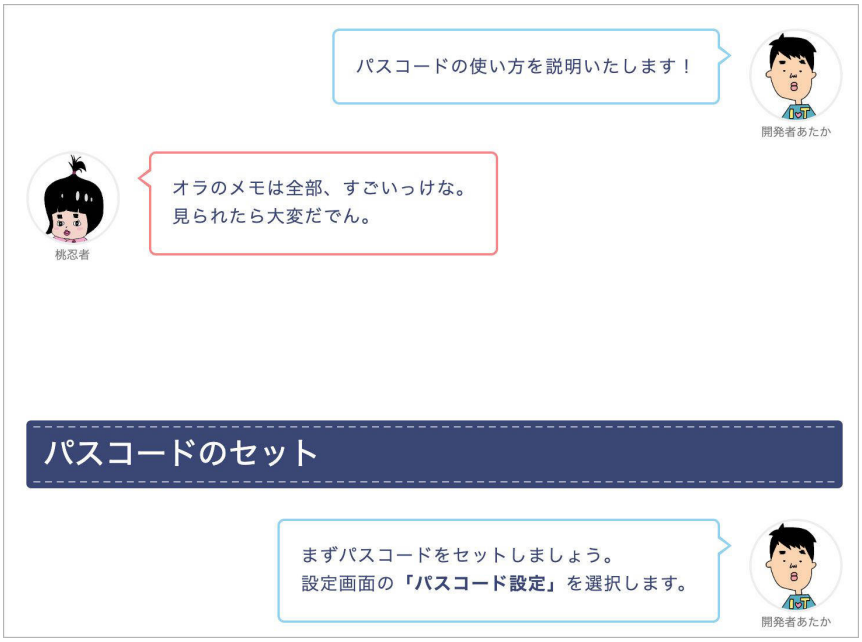
必要なタップ数を1つ減らすだけでも、操作感が劇的に良くなります。徹底的に「もう1タップ減らせないか」と考え抜くのが大事です。

### 操作ガイドと漫画ヘルプ

ユーザー離脱の原因の一つは「操作がよくわからないから」ではないでしょうか。無名の個人アプリであればなおさら、ちょっとでも分からないことがあると、すぐ離脱されてしまいます。

そこで、アプリ起動時には操作ガイドを入れることで、最低限の操作方法が分かるようにしています。また、アプリ内にはヘルプを必ず入れています。ヘルプを読みたくなる工夫として、吹出形式にしたところ、ポツポツと読んでくれる人が増えました！

図9.5: 吹き出し形式のヘルプ画面




テキストだけで一生懸命ヘルプを作ったときは、誰も見てくれませんでした。最近の人は文字をほとんど読まなくて、2人に1人は年間1冊も本を読まない、と聞いたことがあります。だからこそ、徹底的に分かりやすく工夫することが必要です。

## 9.5　オシャレなUIにするためにやったこと

デザインはオマージュで

　僕はデザインが苦手です。頑張って装飾をしては、何度も大惨事になりました。だけど諦めませんでした。

　ついついオリジナリティを出したくなりますが、そこをグッと抑えて、有名アプリのUIを真似しました。色味、行間、雰囲気をスポイトと定規を駆使してオマージュすることで、自分のアプリに取り入れています。

　売れているアプリのUIは1つのスタンダードです。既にユーザーが慣れているUIなら、使い方が分かりやすいので、初期離脱を防ぐこともできます。ちなみに、オシャレなUIカタログアプリのdribbbleは、オマージュの宝庫で、オススメです。

文字色はダークグレー

　プロのデザイナーさんのアウトプットを拝見すると、文字色に黒を使っていないことに気付きました。そこで僕も、文字は黒ではなくダークグレーにして、垢抜けた見た目になるようにしています。

静かなたたずまい

　UIをギチギチに詰め込んで、あれもこれも強調しようとすると、モッサリしたものしかできません。ドロップシャドーを使ったり、文字は気持ち小さく弱くするとプロっぽくなる事に気付きました。一方で、行間や余白は、広めに取るのが良さそうです。スーパーオシャレで洗練されたアプリは、静かなたたずまいです。その感覚を学ぶようにしています。

小さいけど押しやすいボタン

　UIパーツの推奨サイズはiOS:44px、Android:48dp です。確かにそのサイズだと押しやすい反面、見た目が野暮ったくなります。そこで僕は、画像サイズは44pxにしつつ、透明の余白を作ってボタンの見た目を一回り小さくしています。すると、押しやすいけど見た目は小さくてオシャレなボタンができます。

グラデーションは禁止

　僕はシンプルに全部ベタ塗りにしています。iOS7でフラットデザインが一般的になってからは、ベタ塗りデザインが主流になっています。今風のおしゃれな雰囲気になります。グラデーションを活用するのは、素人には難易度が高いと個人的には思います。

ベタ塗りに白抜き

白抜きで文字を描くというテクニックをよく使っています。これで簡単にプロっぽいデザインになります。視認性も良くなります。意識して街なかをみると、アチコチでこのテクニックが使われていることに気付きます。

## 文字は装飾しない

いつもフォントの装飾を頑張るのですが、結局挫折して、シンプルにする形に落ち着いています。文字を装飾すると、なぜかいつもダサくなってしまうのです。シンプルに、装飾しないで、オシャレなフォントを選ぶだけの方が、綺麗に仕上がることが多いです。

## サイレントマジョリティに注意

アイコンがダサくてホーム画面に置けない、とレビューに書かれたので、アイコンを変更しました。すると、他のユーザーから「元の方がかわいくて良かったのに！」と言われて、大炎上しました。大多数の人は不満がなければ声をあげてくれません。レビューやお問い合わせがあると「他のみんなもそう思っている」と勘違いしやすいので注意が必要です。

## デバック式デザイン

オシャレアプリにみんなが慣れているので、デザインを避けては通れません。そこで、僕は「デバック式デザイン」を試行錯誤の中で生み出しました。

頭の中にあるゴールのイメージを積み上げて作っていく「開発型デザイン」が理想です。しかし、デザインが出来ない僕は、こうかなーと作ってみても、何か違和感がある異型のモノを生み出してしまいます。

その異型のモノが生み出された瞬間からデバック式デザインが始まります。UIパーツの位置を変えたり、色を変えたり、大きさを変えたり。1つずつ試しては、1つずつバグを潰していくようにデザインを整えていきます。

デバッグに慣れて勘が良くなったのか、今では最初から納得感のあるデザインができるようになりました。

## 9.6　収益を得るためにやったこと

### 収益モデルは最初に考える

　無料アプリをリリースしたあと、収益を上げるために試行錯誤を繰り返しました。なかなか上手くいかなったので、やむを得ず広告を出したところ、大きな反感を買ってしまいました。

　ユーザーさえ増やせば収益は後から稼げると最初は考えていました。しかし「集まったら簡単」の考え方には落とし穴があります。最初から収益モデルを考えないとダメだと実感しました。

### 月額課金の成約率を上げる

　広告に比べると月額課金はコンバージョン率が低いです。当初は全く課金してもらえませんでした。原因の1つはユーザーの不安感ではないでしょうか。

　そこで「詳しい月額課金説明」「綺麗にデザインされた課金ページ」を用意しました。少しずつ月額課金が成約されるようになってきました。特に「解約方法をしっかりと明記」したのが良かったと思っています。

### 数値やレビューは毎日チェック

　朝起きて、淹れたコーヒーを飲みながら、ストアレビューと数値の確認をすることで、僕は1日を始めます。

　数値やレビューに目を配ると「自分のサービスは大事なものなんだ」という暗示を脳に埋め込めるからです。収益を増やすためにどうしたらいいか、潜在意識からアイデアが出やすくなります。反対に、数値やレビューを気にしないでいると、庭の草と同じで、すぐ荒地になって人が住めなくなるでしょう。

### 納得するまで作り込む

　リリース後の反応は自分の期待の1%にも満たない。僕はそう思っているので、自分が納得するまでリリースしません。絶対売れる！ぐらいのアプリを作って、ようやく「ほどほど売れる」からです。

　適当に作ってみたけど売れたらいいなぁ。イマイチだけど人によってはウケるかも。そういったアプリが売れたことは、今までの経験で一度もありませんでした。リリース後に反応を見て改善しようにも、そもそもの反応が無いのでお手上げです。

## 9.7　安全で効率的な開発のためにやったこと

端末代をケチらない

　実機でアプリを触わりまくって、操作の違和感を必ず全部潰すことが大事です。たまにサボると、その度にひどい目にあっています。シミュレーターだけで済ますのは言語道断！特徴的な新端末が出た時は必ず買って、表示だけじゃなく操作感も確認しています。

テストのための「本番」環境

　人気が出てきたメインのアプリだと、新しい機能を入れるのはリスキーです。そこで、アクセスが多くないセカンドサービスで新機能を試すことにしています。ユーザーの反応を見ながら新機能を改善して、安定した所でメインサービスに投入します。こうすることで安定的に新機能を追加できます。

課金だけ先に審査を通す

　アプリをリリースするには、AppleやGoogleの審査を通さないといけません。特に課金に対するチェックは厳しいため、初回審査には時間が掛かることもしばしば。

　そこで僕は「最低限の機能」+「課金処理の完全実装」で審査をまず通しています。その上で、完成版を再審査に通すことで、待ち時間を短縮しています。盛り上がった熱が冷めないうちにリリースできます。

汎用機能を積み重ねる

　他のアプリで使えるよう、機能は汎用的に作っています。最初は1ヶ月ぐらいかけて作った機能が、次のアプリでは数分で追加できるのです！前のアプリでテストできているので、安定動作も保障されています。

　課金、レビュー、お問い合わせ、プッシュ通知等々をモジュール化して、次の開発工数をぐっと削減しています。この汎用機能の積み重ねによって、個々のアプリの作り込みに十分な時間が取れるようになりました。

## 9.8　プロ個人開発者の働き方

開発にはMacBook

　周りの個人開発者はMacBook Proを勧めますが、僕は軽くて薄いMacbookを愛用しています。スペックは十分高いので、Photoshopをいじりながらでも、XCodeもAndroidStudioもスムーズに動きます。何より920gという軽さは、膝の上でさえ気軽に開発できて、取り掛かるハードルが低くなるのが大きなメリットです。会社員時代は自分のMacbookを持ち歩いて、休憩時間に個人開発をしていました。

小さな習慣

　仕事で疲れて帰ってきてから、しっかり2時間、個人開発が出来るか？きっと「無理」です。

　でも、家でパソコンの電源を入れるだけなら出来るかも。エディター立ち上げるだけならできるかも。関数を1つ作るだけならできるかも。毎日1分ならできるかも。それなら、5分は？10分は？

　こうした小さな一歩を積み上げると、個人開発が習慣になります。「無理」だったはずが、気づいた頃には「当たり前」になります。

　会社員時代は、早起きをして、朝に開発をする時間を必ず設けました。そこから少しずつ開発時間を増やせるように頑張ってきました。

ドコデモ開発～

　どこでも気軽に開発を始められる「ドコデモ開発」は最強の習慣です。個人開発を成功させるためには、アイデアとか技術の前に、圧倒的な手数が必要だと考えています。

　個人開発を専業にすると、何の強制力も働きません。黙々と家で作業を進めて、やがては習慣になりました。すると、帰省の新幹線の中でも、気軽にコーディングが出来るようになりました。

## 9.9　プロ個人開発者の考え方

リスクを減らす

　会社員時代に「起業資金」という名目で毎月貯金をして、無収入でも5〜6年は生活できるバッファーを用意しました。それに妻が正社員としてフルタイムで働いていて、子供もおらず、借金もありません。ダメだったら再就職するという気持ちも残して、破産リスクを限りなくゼロにしました。

　借金を背負って、土日深夜と頑張って大成功する話は、たしかにカッコいいですよね。だけど、個人とその家族が生きていく稼ぎを作るのに、そんな大挑戦は必要ないんじゃないかな、と思っています。

諦めない

　成功のコツは「諦めないこと」だと実感しています。

　ユーザーからの反応が薄かったり、ダウンロードが下がってきたりすると、ネガティブになることもあります。「やっぱり個人アプリで食べて行くなんて無理なんだ」とやめたくなってしまいます。

　でも、不安の原因は、根拠の無いただの思い込みです。いつも心を強く保つ秘訣は「根拠のない事は良いように考える！」です。

　誰かが出来てるなら、自分にもできます。何かを見て良いと感じられる感性があるなら、同じものを自分でも作れます。限界を決めるのは、自分の能力じゃなくて「出来ない」という思い込みです。「正しい」思い込みがあれば、何でもできるはず、と信じて毎日頑張っています。

スタート地点

　どこを見ても、大人も子供もお年寄りも、みんなスマホを持っている。なのに、なぜ自分のアプリはダウンロードされないんだ！？お金を儲けるのって、物凄い難しい！！儲かってる会社って実は超凄い！！

　この現実を理解することが、個人開発で得られる一番の価値だと思っています。そこからやっと、スタート地点に立てます。これは本当にやってみないとわからない感覚です。

全部やる

　僕のようなスタイルの個人開発では、大きな何かがドーンと結果を出すわけではありません。ひとつひとつの小さな努力の積み重ねが、結果に繋がるのだと思っています。

　だから「1%でも成功率が上がるなら必ずソレをやる」と心掛けています。やった方がいいと思ってるのに、やらない。やらない理由が「面倒くさい」なら、歯を食いしばってやる！絶対やる！！

悩みを解決するシンプルな方法

悩みの9割は「つべこべ言わずに行動しろ」で解決するらしいです。

・ダウンロードが少ない → コード書け

・ヒットさせたい → コード書け

・収益が上がらない → コード書け

・凄いコードが書けない → コード書け

迷ったら、うじうじ悩まないで、とりあえずコードを書くようにしています。

今日やらないことは明日もやりません。今日までの人生でイヤというほど分かっています。「someday island」（いつかやる病）です。

時間がない？技術がない？お金がない？着る服がない？出来ない理由を考えるのではなく、出来るようにする方法を考えることに全てを向けましょう。

と、物凄い面倒くさがりでサボリ症の自分の気持ちを毎日奮い立たせています。[5]

```
アタカ / @atagon


個人で作ったサービスだけで生活をする「プロ個人開発者」を自称。
脱サラ・受託0・売上月40万円を達成。毎月の売上をnoteで公開している。
「付箋todoメモ帳 QuickMemo+」等のアプリを開発。


https://apps.apple.com/jp/app/id513272765
```

1. https://apps.apple.com/jp/app/id513272765
2. https://apps.apple.com/jp/app/simple-quicktodo/id845111209
3. https://apps.apple.com/jp/app/id1326572969
4. https://apps.apple.com/jp/app/id955088460
5. 本稿は執筆者（@atagon）によるブログ記事「個人開発 虎の巻」シリーズを加筆・修正して寄稿したものとなります。

# 第10章　ストアからアプリが削除！消された個人開発アプリとその理由を公開！

　こんにちは。ネットビジネスで生計を立てているSaaay（@pompompomer）です。個人開発を始めて5年間、ストアからアプリを削除されてしまうなど、危うい場面が幾度かありました。そんな失敗談の数々をご紹介します。これからアプリを開発する人の反面教師になれれば幸いです。

## 10.1　アプリの開発歴や実績はどのくらいあるか

　初めてアプリを開発したのが2013年の秋頃です。時にWeb開発をしてみたり、IT企業で働いてみたりをしながらも、なんやかんやで執筆時（2019年）現在まで惰性的に続けています。

　これまで開発してきたアプリは、非公開のものも含めれば100タイトル、累計ダウンロード数は850万になります。主にAndroidアプリを中心に開発をしています。

## 10.2　これまでどんなアプリを作ってきたのか

　以下に代表的なアプリを2作挙げてみます。なお、DL数は各アプリの公開日から2019年1月現在までの累計となります。

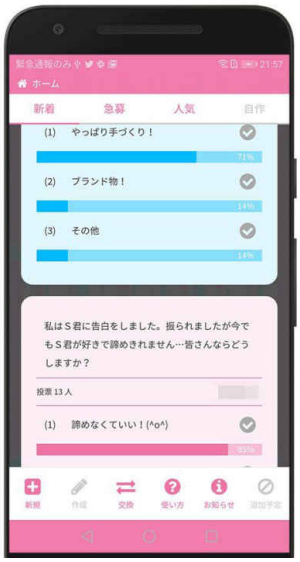
「付き合える度診断」：約80万DL：評価4.0

図10.1: 付き合える度診断


　好きな人と付き合える確率を算出する「付き合える度診断」[1]です。確率は対象の人との距離感や親密度などをもとに算出しています。

図10.2: 付き合える度診断 - サービス概要画面





また、不定期に更新される恋愛コラムやユーザーが答えてくれる恋愛アンケート機能など、恋愛に関するサポートツールも充実させています。

「精神年齢診断」：約70万DL：評価3.8

図10.3: 精神年齢診断


自分の精神年齢を計測する「精神年齢診断」[2]です。診断結果としては精神年齢の他、性格要素のレーダーチャートが表示されます。

図10.4: 精神年齢診断 - サービス概要画面


また、付録として全国ランキング付きのパズルも用意しています。

その他の開発アプリ

脚注にURLを掲載しています。ここから公開中の全アプリを確認することができます。

・現在活動中のアカウント[3]

・開発を始めた初期の頃のアカウント[4]

## 10.3　どんなアプリを消されてしまったか

　これまでどんなアプリが、どんな理由で警告を受けたり制裁を受けたりしたのかを、実際の警告メールとともに紹介したいと思います。

## 10.4　事例1：[警告]コンテンツレーティングが合っていない

　コンテンツレーティングとはアプリの内容がどの年齢層を対象にしているかを決める評価です。これが不適合との指摘を受けた2014年の事例を紹介します。この時は、強制的にレーティングを変更されました。

　以下がその報告メールの全文となります。

```
Google Play チームによる定期的な審査の結果、アプリ ツンデレ度診断
 – わたし、ツンデレかも！？（パッケージ ID com.lastprojects111.tsunderetest）の
最低コンテンツ レベルが Low Maturity (2) に変更されましたのでお知らせいたします。

コンテンツ レベルの変更の理由： Google Play コンテンツ レーティング ポリシーへの違反

コンテンツ レーティングがこれ以上変更された場合は、違反しているそれ以外の
アプリの削除を含め、管理上の措置をとらせていただく場合があります。
違反はすべて追跡されます。深刻な違反があった場合や、違反を繰り返した場合は、
それがいかなる性質の違反であってもデベロッパーアカウントが停止され調査が行われます。
また、関連する Google アカウントが停止される場合もあります。
Google Play ユーザーに正確なコンテンツ レーティング情報を提供できるよう、
ご協力のほどよろしくお願いいたします。
```

　指摘を受けたのは、未だにストアで販売している『ツンデレ度診断』というアプリです。

　このアプリは今も昔も、卑猥な要素や暴力的な要素を含めていません。おそらく、広告に何か良からぬものが含まれていたせいで、違反の対象と見なされたのではないか、と推測しています。

　それにしても「アカウントの停止」と言及されると、身震いします……。

## 10.5　事例2：[警告]プライバシーポリシーがない

次に紹介するのは、2017年2月に指摘されたプライバシーポリシーの問題です。この時期に行われたデベロッパーポリシーの更新に伴い、広告やアナリティクスなどにより特定の権限が必要なアプリにはプライバシーポリシーの表示が必須となりました。

以下がその指摘を受けた際のメール全文です。やや長いので、さらっと流し見てください。

```
お客様のアプリ マイペース診断（パッケージ名 net.testii.mypacetest）が、
Google Play の個人情報や機密情報に関するユーザー データ ポリシーに違反していることが確認されました。

警告の詳細： Google Play では、ユーザーや端末に関する機密情報を要求する、
または取り扱うアプリの場合、デベロッパーは有効なプライバシー ポリシーを提供する必要があります。
当該のアプリは、個人情報または機密情報に関わる権限
（カメラ、マイク、アカウント、連絡先、スマートフォンなど）
またはユーザー データを要求していますが、有効なプライバシー ポリシーが確認できませんでした。

必要な対応：［ストアの掲載情報］ ページとアプリ内に有効なプライバシー ポリシーへの
リンクを記載してください。詳しくは、こちらのヘルプセンター記事をご参照ください。

なお、機密情報に関わる権限やユーザー データに対するリクエストをアプリから
すべて取り除かれた場合には、上記のご対応は必須ではございません。
また他にも公開している同様のアプリがある場合、すべてのアプリについて
目立つ方法での開示の要件を満たしているかどうかご確認ください。
2017年3月15日までにこの問題を解決いただくようお願いいたします。
ご対応いただけない場合、Play ストアからの削除も含め、
お客様のアプリの公開を制限する措置を取らせていただきます
恐れ入りますが、あらかじめご了承ください。
Google Play のユーザー皆様にわかりやすく透明性の高いサービスを提供するため、
ご協力いただけますと幸いです。どうぞよろしくお願いいたします。
```

先ほどの「アカウント停止」の内容に比べて、すごく柔らかい印象の文面です。もちろん、素直に従い、全アプリにプライバシーポリシーを用意しました。

なお、現在のデベロッパーポリシーにおいては、プライバシーポリシーは、Google Play Console にリンクを記載するだけでなく、アプリ内にも開示する必要があります。

## 10.6　事例3：[削除]イラストの露出度が高い（Part1）

　次はアプリ自体が削除された事例です。「遊び人診断」というアプリです。このアプリはメインキャラクターとして、やや露出度の高いキャラクターを用いていたため、指摘を受けました。

図10.5: 遊び人診断


　Google先生はいかなる谷間も見逃しません。以下のようなメールが届きました。

```
Metadata: “We don’t allow apps with… excessive, or inappropriate metadata,
which include the app’s description, title, icon, screenshots, and promotional images.”

Please adjust your app’s description, title, icon, screenshots,
and promotional images to make sure your app is policy compliant and eligible
for promotion. We’ve indicated the flagged content in your app listing below.

App: 遊び人診断 (net.testii.asobinintest)

Flagged content: icon, screenshots, feature graphic

Issues:

Real-life or fantasy humans/creatures portrayed in clothing that provides
excessively tight, or minimal coverage of breasts, buttocks, or genitalia

Real-life or fantasy humans/creatures portrayed in sexually suggestive poses
```

なぜ届くメールが日本語だったり英語だったりするのか気になるところです。ここで指摘されているのは「メタデータに関する問題」のようです。

つまり、ストアで表示されるイメージ（アイコンやスクショ、プロモーション画像）に関する問題であり、アプリ内のコンテンツに対する指摘ではないということ。

アプリ内のコンテンツは、レーティングを高めることで、提供するユーザーを制限することができます。一方で、ストアの商品紹介の画像は、年齢を問わず、すべてのユーザーが目にすることが可能です。それゆえ、これらに関しては、より厳しく審査される模様です。

## 10.7　事例4：[削除]イラストの露出度が高い（Part2）

先の「遊び人診断」の指摘を受けたのは2016年のことです。それ以後、自分は露出度の高いキャラクターの使用を一切やめました。

しかし、2019年1月、突如ひとつのアプリが同様の問題で削除されました。2014年に公開以後、30万DLにいたるまで、一切問題なく公開されていた「RPG適職診断」というアプリです。

その時の先生からのメール、および添付されていた画像は以下になります。

> 最近のレビューの結果、RPG適職診断 – 旅立つ前に己の役を見極めろ！
>
> Testiiの診断・心理テストシリーズ（net.testii.rpgtest）はGoogle Playから削除されました。
>
> 公開ステータス： 削除済み
>
> お客様のアプリは、ポリシーに違反していると判断されたため、削除されました。
>
> ポリシーに準拠するアップデートをご送信いただくまで、
>
> このアプリはユーザーに配布、販売されませんのでご了承ください。

図10.6: RPG適職診断のストア画像（谷間あり）


添付画像『RPG適職診断』

この左上の僅かな谷間……これがダメだと仰るのです。なるほど、確かに現実にこのような服装の人が歩いていたら、露出が高いと判断されるかもしれません。が、少年ジャンプの表紙だって、ゲームのパッケージだって、なんなら秋葉原を歩けば……という感情が湧いてきます。

図10.7: RPG適職診断のストア画像（谷間なし）


修正後画像『RPG適職診断』

しっかり胸をロゴで隠してすぐに申請を出しました。すると、数時間後には販売が再開されました。

## 10.8　事例5：[削除]広告の表示方法が望ましくない

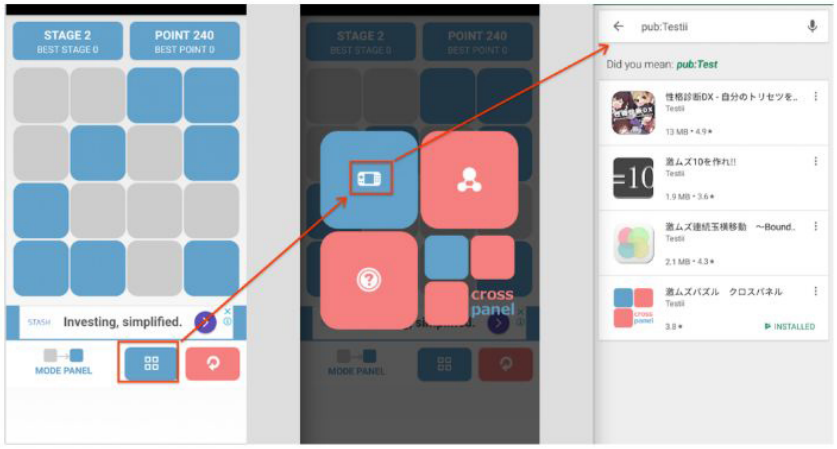
最後は自社広告に関するもので、上記と同じく、2019年1月に指摘された問題です。

アプリ内に自分のアプリの一覧ページへのリンクを設けていました。おそらく、そのボタンが広告だとは判別しにくかったのが原因だと思います。

その時のGoogle先生からのメール、および添付されていた画像は以下になります。

```
最近のレビューの結果、激ムズパズル　クロスパネル（net.testii.crosspanel）は
Google Playから削除されました。

公開ステータス： 削除済み

お客様のアプリは、ポリシーに違反していると判断されたため、削除されました。
ポリシーに準拠するアップデートをご送信いただくまで、このアプリはユーザーに配布、
販売されませんのでご了承ください。
```

図10.8: 激ムズパズル クロスパネル の広告




添付画像『クロスパネル』

言いたいことはわかります。

でも、診断アプリばかり出している自分にとって、数少ないパズルアプリです。なにも削除まですることは……。

そう思いながらも、これから対応予定です。今はほとんどダウンロードされなくなったアプリだったので、急がず焦らず、手が空き次第対応したいと思います。

## 10.9　Google先生に怒られないようにするためには

以上がこれまで自分がGoogle先生からいただいたお叱りや制裁でした。

これらを省み、Google Playでアプリ公開する際に注意する点をまとめると

・広告を表示するなら、レーティングをやや高めに設定しておく。どんなにフィルタリングをしても、露出度の高い広告が含まれる可能性は否めない。

・アイコンやスクショ、プロモーション画像をはじめとするストア表示画像では、露出の高いキャラを避ける。

・プライバシーポリシーはきちんと用意する。

・自社広告と言えど、アイコンボタンだけではだめ。宣伝であることを明記する。

のようになります。Google Playに限らず、プラットフォームに乗るときは、プラットフォームの規約に配慮しないといけませんね。

なお、デベロッパーポリシーは日々更新されていくものなので、新しい規約が適用された際には随時対応していく必要があります。

皆様が自分のようにアプリを消されることがないことを願っています。お読みいただき、ありがとうございました。[5]

```
Saaay / @pompompomer

高校退学→大検取得&大学卒業→アプリを作り生計を立てている個人開発者です。
『動物キャラ診断』『付き合える度診断』など、全アプリの累計DL数は850万、
月次最高売上は160万円。大阪を拠点にノマドワーカー＋時々釣り。

https://play.google.com/store/apps/developer?id=Testii
```

1. https://play.google.com/store/apps/details?id=com.lastprojects111.tsukiaerudotest&hl=ja
2. https://play.google.com/store/apps/details?id=net.testii.mentalagetest
3. https://play.google.com/store/apps/developer?id=Testii
4. https://play.google.com/store/apps/developer?id=LastProjects
5. 本稿は執筆者（@pompompomer）によるブログ記事『アプリ開発やネットビジネスの情報を発信するブログを始めました！』『ストアからアプリが削除？消された個人開発アプリとその理由を公開！』を加筆・修正して寄稿したものとなります。

# 第11章　自分の漫画ライフを充実させる「新刊通知アプリ」を作ってみた

　僕（@toiroakr）は、「Comic Trace」[1]というアプリを開発しました。会社ではエンジニアとして働いていますが、個人開発は今回が初めてです。上手くいったことも、そうでなかったことも沢山あります。初の個人開発での苦労と学びを振り返ってみたいと思います。

## 11.1　どのようなアプリを作ったのか

「Comic Trace」は漫画の新刊が発売された時に通知してくれるアプリです。

図11.1: ホーム画面


ホーム画面：アプリを起動するとこの画面が開き、発売予定日の近い順で漫画を表示します。画面の更新タイミングは、アプリ起動時、更新ボタン押下時、登録変更時です。

図11.2: 登録画面


登録画面：漫画のタイトルを入力→検索→登録ボタンを押すと、その漫画の新刊情報がホーム画面に表示されます。検索結果の下には登録済みの漫画が一覧表示されており「解除」ボタンを押すことができます。

設定画面：「いつ通知を受け取るか」といった設定ができます。

## 11.2　なぜ作ったのか

既存アプリに対する期待と不満

僕は漫画が好きで、現在は40タイトル以上を継続的に購入しています。どのタイトルの新刊がいつ発売するのか覚えるのは現実的ではありません。

そのため、新刊通知アプリの存在を知ったときには、期待を抱きました。しかし、既存のアプリをいくつか試したのですが、以下のような不満が湧きました。

①スピンオフ作品の通知が来る。例えば「進撃の巨人」には「進撃の巨人 Before the fall」「進撃の巨人 悔いなき選択」「進撃の巨人 Lost Girls」といったスピンオフ作品があります。多くの新刊通知アプリで「進撃の巨人」を登録すると、これら周辺作品の通知まで受け取ることになります。人によっては嬉しいかもしれないですが、僕としてはあくまで別で管理したいところです。

②電子版の販売日に対応していない。コミックス版（紙媒体）をメインにしているアプリが多いようです。そのため通知を受け取って新刊を買おうとしても、電子版はまだ発売してなかったりします。僕はKindleユーザーなのでちょっと不便です。

③新刊情報が一目で分からない。新刊通知の機能だけでなく、購入済みの漫画を管理できるような、リッチなアプリが多いです。それゆえに導線が複雑だと感じました。カレンダーから日付を選択しないといけなかったり、何度かタップやスクロールしないと、新刊情報に辿り着けません。

④広告が邪魔。漫画を追加する度に画面全体に広告が出てくるようなアプリもありました。せめてバナー広告にしてくれないかな、ちょっとつらいな、と思いました。

開発者としての好奇心と焦り

以前から個人開発への憧れはありました。しかし、日々の仕事や技術インプットを言い訳にして「アプリ開発」には手をつけてはいませんでした。そんな僕が個人開発に挑戦して、こうしてリリースに至ることができたのは、3つの理由があったと思っています。

ひとつ目は、身近な個人開発者の存在です。個人開発をしている同僚とランチに行く機会がありました。溜め込んでいるアイデアや今作っているサービスの話を聞いていると、自分でもやってみたいなという気持ちが高まりました。

ふたつ目は、技術面での好奇心です。フロントエンドの仕事でTypeScriptを使う機会が増えました。そこで「バックエンドもTypeScriptで書きたいな」「新しいフレームワークを使ってみたいな」という気持ちがありました。せっかくならこの機会を活かしてプライベートでアプリを作りながら学ぼうと考えました。

三つ目は、アウトプットへの焦りです。エンジニアがWebに公開しているアウトプットを評価して、企業のスカウト採用をサポートするサービスが次々と登場しています。これまで以上にアウトプットの重要性が増していると感じました。社内で成果を挙げるだけではエンジニアとして不十分ではないかと焦り始めました。

## 11.3　どのように作ったのか

技術選定：アプリ編

TypeScriptを使う場合「React」「Vue」「Angular」といったライブラリーが有名です。今回はReact Nativeを選択しました。補助サービスであるExpoを併用しています。これらのおかげでReactでiOSやAndroidのアプリを開発することができました。

当初は、Weexというフレームワークに興味がありました。Vueで似たようなことができるとのことですが、いざ試してみるとハマりどころが多かったです。例えば、一部のプラグインを導入するには、中国語のドキュメントを読まないといけません。

Angularでモバイルアプリを開発できるフレームワークがionicです。ionicは仕事で使っているので、個人開発では別の技術を使おうと思いました。

Flutterというフレームワークもあります。同じようにクロスプラットフォームアプリを開発できるそうです。これも気になってはいましたが、TypeScriptではないので、今回は見送りました。

結果的にはReact Native + Expoを選んで良かったと思っています。前職でReactを使ったことがあるのでスムーズに開発を始めることができました。

技術選定：サーバ編

サーバーサイドAPIには「typescript」「framework」でググって一番最初に出てきたNestJSを採用しました。漫画情報の取得はAmazonのProduct Advertising APIを使用しています。後述しますが、漫画情報の取得に関してはRubyのライブラリーに頼りました。

インフラは紆余曲折を経てGCP（サーバー）とMongoDB Atlas（DB）に移行しました。

最初はHerokuで完結させるつもりでした。Herokuは仕事で使っているので知見がありました。何より無料で構築できるのが魅力的です。しかし、開発を進めるうちに、問題が浮上しました。

まず、サーバーのスリープ問題です。Herokuの無料枠では30分間アクセスがないとサーバーがスリープします。システムで定期的にアクセスを発生させて、スリープを防ぐように設定しないといけません。なんだかなぁと思って、最終的にはGoogle App Engineをサーバーとして使うことにしました。漫画情報の取得はGoogle Compute Engineで動かしています。

次は、データベースの容量制限です。HerokuのPostgreSQLを使ったのですが、無料枠には1万行の容量制限があります。実際にデータを入れてみると無料枠を維持するのは厳しそうでした。そこで、代わりに無料枠の容量が多いMongoDB Atlasに切り替えました。

## 11.4　どのような苦労があったか

漫画タイトル・巻数の取得

AmazonのAPIは『進撃の巨人（２６）　（週刊少年マガジンコミックス）』といった商品名を返します。ここから漫画の「タイトル」や「巻数」を抽出する必要があります。

最初は正規表現でパースしようと頑張りました。しかし上手く行かず「このままだと無限に終わらないのでは……」と困ってしまいました。そんな折に、本書の共同執筆者でもあるmorizyunさんに「amakanize」というRubyのライブラリーを教えてもらいました。amakanizeの完成度は高く、多くの漫画で期待した情報を抽出できるようになりました。

amakanizeで対応しきれないケースもありました。泥沼にハマりそうだったので余計なことはせず、対症療法的に修正・対応しています。以下が例です。

①『地獄先生ぬ～べ～』が『地獄先生ぬ（べ）』になってしまう問題。ライブラリーの仕様では「～hoge～」が「（hoge）」に変換されてしまうようです。おそらく『漫画タイトル ～サブタイトル～』形式に対応するためだと思います。変換結果が『地獄先生ぬ （べ）』に一致した場合だけ『地獄先生ぬ～べ～』に修正するように処理を差し込んでいます。

②「電子版」「コミックス版」問題。Kindle版の漫画の多くは、タイトル以外の付加情報が付きます。『漫画タイトル ＜電子版限定カラーイラスト収録＞』といったように。そこで、紙（コミックス）を優先して採用するようにプログラムを書きました。紙は紙で『漫画タイトル ＜コミックス版＞』といったケースもありましたが、Kindle版に比べるとパターンは少ないので、正規表現で対応しています。

実際のデータをひとつひとつ見ながらこういった問題を解決しました。おそらく100%を修正できたわけではありません。今後も新しいケースが発生するかもしれません。Amazon側で形式を揃えてくれないと、完璧に対応するのは無理です。

他の漫画新刊通知アプリでは、スピンオフ漫画まで通知が来るようになっています。こういった地道な苦労を避けるためなのかなと思いました。

Expoとアプリのビルド

「全部Expoに任せてOKだよ！」みたいな感じだったので、特に何も考えずに開発を進めました。ところが、もうすぐアプリが完成するぞというタイミングで、技術的な問題が浮上しました。

①「プッシュ通知が送れない」問題。Expoのプレビューアプリでは、問題なく通知が成功しました。しかし、TestFlightで配布したiOSのベータ版では、通知が届きませんでした。

調べていくとアプリをビルドする際の設定に原因があることが分かりました。「Provisioning Profile」という設定の「Enabled Services」にプッシュ通知が含まれていないようです。

Expoのサポートフォーラムで質問して、以下の対応手順を案内いただきました。

１．初回ビルド時に作られたApp IDのプッシュ通知の欄を「Configurable」に変更する。

２．初回ビルド時に作られたApp ID以外のProvisioning Profileなどを削除する。

３．expo build:ios --clear-credentials コマンドで再ビルドを実行する。

この手順に従って無事に解決しました。

②「WebViewが表示されない」問題。アプリ内ブラウザでWebページを表示するのに、ExpoのWebViewを使っています。iOSでは何の問題もなく動作したので「アプリが完成した！」と思っていました。ところが、AndroidでWebViewを開くと「about:blank」が表示されてしまいました。

アプリ内ブラウザを使うのは諦めて、常に外部ブラウザで開くように仕様を変えました。正しい修正方法は調べてさえいません。面倒臭いなぁと思って、諦めてしまいました。「せっかくReact Nativeで作ったのだから、ついでにAndroid版もリリースするか」くらいの気持ちでした。

いかに便利なツールであっても、技術的な問題はついて回るようです。自分なりに調べたり、サポートに問い合わせたり、仕様を変更したりと、対応力が求められました。

## 11.5　どのような学びがあったのか

仕様について

僕はたまにComicTraceを開いて、予約可能になっている漫画があれば、その場で予約しています。自分の不満をもとに開発したので、元々あった不満については解消できました。ところが、自分でアプリを使っていると、段々と新しい不満が出てきました。

①「既に予約済みだった」問題。どの漫画を購入予約したか、わざわざ覚えていません。いざ予約しようとすると、既に予約済みだった、ということが何度も起きました。わざわざリッチな購入管理機能を作る必要はありませんが、せめて「閲覧済み」マークを付けると便利になりそうです。

②「バッジが邪魔」問題。漫画の発売日には通知が届きます。既に予約は完了しているので、ComicTraceを開く必要はありません。Kindleで漫画を読めばOKです。ところが、アプリのアイコンには通知のバッジ（未読マーク）が残ります。地味ですが、不満です。「発売日当日」の通知ではバッジを付けないほうが満足度は高そうです。

③「当日0時に通知してほしい」問題：設定した日付（当日や前日など）の午前8時半に通知が届くように作りました。Kindle版は当日の深夜0時に新刊が配信されるので、当日0時の通知も追加したいなと思うようになりました。

自分のために個人開発したはずなのに、問題だらけです。不満が出ないものを作りきるのは難しいのだと思いました。

開発について

React Native、Expo、NestJS、MongoDB、いずれも今回が初の利用です。何かしようとする度にドキュメントを読む必要があって大変でした。個人開発の経験を積むうちに、自分にとってベストな技術スタックが定まるのだろうなと思います。

苦労話も書きましたが、Expoを選んだのは正解でした。expo publish コマンドを実行するだけで、OTA（ワイヤレス）アップデートも可能です。開発者としてはアプリの審査レビューを待つ手間が減ります。ユーザーとしてはストアからダウンロードしなくて済みます。すごく楽でいいなと思いました。

一方で、リリース後には、また違った苦労がありました。新機能を追加したわけでもないのに、メンテナンス作業が発生します。例えば「タイトル・巻数の取得」処理は、たまにプログラムを修正しています。Amazonの商品名が更新されたり、新しいレーベルや作品が登場すると、上手くパースできないことがあるからです。

最初は作った後のことまで考えていませんでした。僕が個人開発に割ける時間はそこまで多くありません。メンテナンスが大変だと、サービスの提供継続や機能改善、他のサービス開発に手が回らなくなります。

できるだけ早いタイミングから、運用負荷を減らせるような仕組みを考えるべきだったと思います。問題の発見・修正を容易にしたり、そもそも問題が起きないようなシステムを目指したいです。

ユーザーについて

苦労は多々ありますが、おおむね満足しています。僕は自分の漫画ライフを充実させるために、ComicTraceを使っています。自分自身をターゲットユーザーに設定したことは正解でした。

いざ「個人開発をしよう」と思っても、すぐには作りたいものが思いつきませんでした。「自分は何が作りたいんだろう？」と考えても、なかなか答えは見つかりませんでした。だけど視点を変えて、自分の生活の中での不満を探したら、すぐにComicTraceを思いつきました。僕にとっては「この不満を解消するにはどうすればいいだろう？」と考える方法が、向いていたようです。

要件を決めやすいというメリットもありました。自分のニーズに注目すれば良いからです。ニーズが弱いと諦めてしまいがちですが、途中で「違うな」と思ったら方向転換しても構わないでしょう。そう思うと気楽にアプリを作れます。

他人が求めているものを想像しながら開発するのは難しいなと思います。スタートアップの業界には「ユーザーの声を聞くな」という考え方があるそうです。自動車を知らない時代の人に、何が欲しいかと尋ねると「より足の速い馬が欲しい」と答えてしまう。その言葉を真に受けていたら、自動車は発明されなかっただろう。ユーザーの言葉の奥にある本当の課題を想像して、その課題を解決するための製品を作ろう。そういった考え方です。

かといって、想像だけでアプリを作って、誰にも使われないというオチになると、ちょっと悲しいです。なので、自分のためにアプリを作って、少なくとも自分1人はそのアプリを使う。このやり方が僕には合っていました。初めて個人開発に挑戦する人には、自分のためのアプリを作ることをオススメします。アイデアを他の人に話して共感を得られるかぐらいは確認しても良いかもしれませんね。


ひぐち / @toiroakr

大学院で情報学を専攻後、大手企業でEdTechサービスを開発。

現在は創業間もない教育系スタートアップで働くソフトウェアエンジニア。

個人開発した漫画新刊通知アプリ「ComicTrace」は諸事情で公開停止。

https://github.com/toiroakr


1. 諸事情により記事執筆から本書発売の間にアプリの公開を取りやめています。

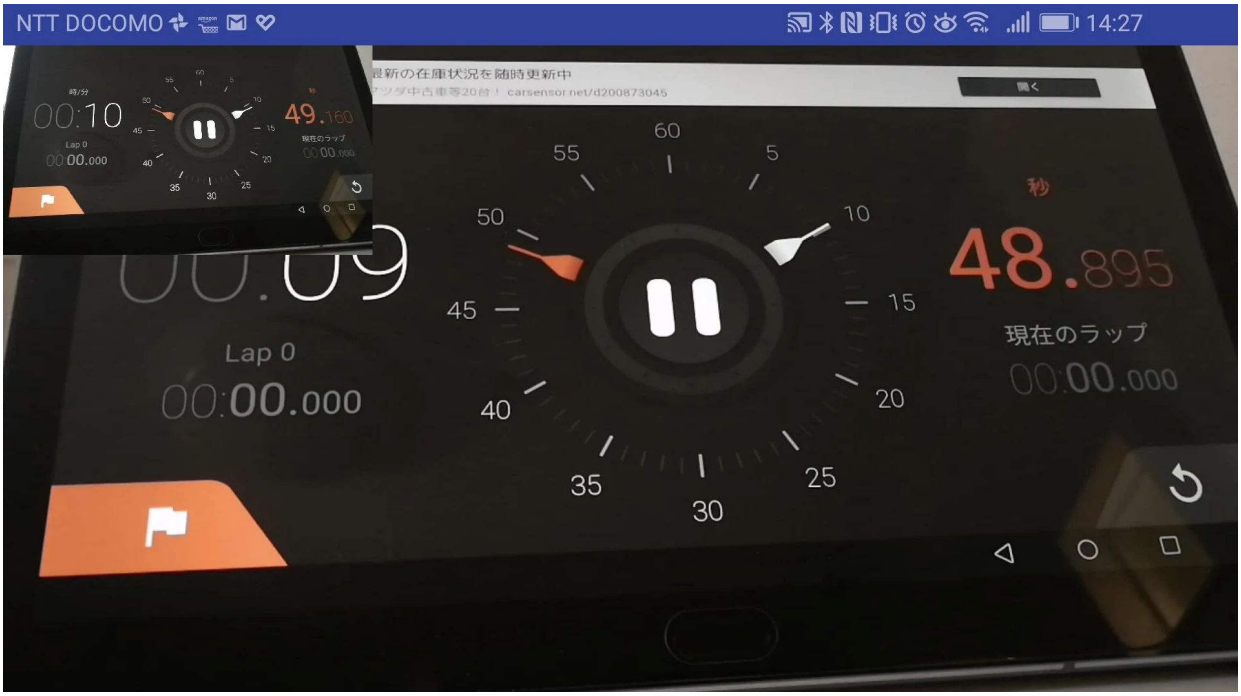
# 第12章　カメラアプリを開発して憧れのレビューサイトに掲載された話

　個人開発者の @kohshin1977 です。初めて「アプリを作っています」と自慢できるようなアプリを無事にリリースできました。憧れだったレビューサイト掲載と、その効果を振り返ってみたいと思います。

## 12.1　遅延再生カメラ

　遅延再生カメラAndroidアプリ「遅カメ」（ちかめ）[1]を開発しました。カメラからの映像を1～120秒遅延させて表示させることができます。ボルダリング、スラックライン、その他のスポーツのフォーム確認に最適です。

図12.1: アプリ画面




　遅延された映像をもう一度見るプレビュー機能を搭載。プレビューで表示した映像をmp4として保存可能。RAMを使用するため、Android端末のフラッシュメモリーの寿命が縮むことはありません。

遅カメを作ったきっかけ

　作者はボルダリングというスポーツをやっています。ボルダリングとはロッククライミングの一種で、壁に取り付けられたホールドを頼りに壁を登るスポーツです。最近はボルダリング用のジムが増えており、実際に体験された方もいるのではないでしょうか。

　自分もジムで登っていて、たまにスマホで撮影するのですが

　１．録画開始ボタンを押す

　２．登る

　３．停止ボタンを押す

４．動画を再生する

という動作をしなければならず、面倒臭い思いをしていました。

遅延再生させてやれば上記の動作を省略できるのではないか。そう考えたのがこのアプリを作ったきっかけです。

フォーム確認が必要なスポーツなら幅広く使えると思います。何度もトライが必要な物事で、成功した場面だけ録画したい場合に有効です。

## 12.2　レビューサイトでの紹介

「遅カメ」はAppliv[2]というレビューサイトで紹介されました。

> 用途がかなり限られているが、いざという時には便利。

若干のイジリもありながら、なかなかの褒め具合です(笑)。レビューサイトに載せてもらうことは目標のひとつだったので大変嬉しいです。

図12.2: アプリランキング

フォームチェック・練習用ビデオカメラ アプリランキング

そしてなんと！Applivの「フォームチェック・練習用ビデオカメラ アプリランキング」で2位に浮上しました。まあ、同じカテゴリは6個しかないんですけどね(笑)。

ただ、ちょっと失敗したことが……。アプリストアの画像がそのまま使われていたので、もっとちゃんとした画像をアップしておけばよかったなぁと思いました。

アプリストア用に画像を用意するのは面倒臭いんですよね。最初に試しでアップロードした画像のままで、公開してしまいました。ということで、レビューサイトに申し込むときは、アプリストアの画像もベストな状態にしておきましょう。

## 12.3　ダウンロード数は……

　レビューサイト掲載から2週間でダウンロード数がどう推移したのかを見てみましょう。

図12.3: インストール数


　何も変わりませんでした…。5/13日にApplivで紹介されてからも、ダウンロード数は10近辺をうろうろしてます。

　なんてこった……！レビューサイトで紹介されればダウンロード数は伸びると思っていたのに。個人開発者はどうやってダウンロード数を伸ばせばいいんだ……！？

　ん？ちょっと待てよ。

　Play Consoleのユーザ獲得を見ると、意外なことが分かりました。

図12.4: 4月29日～5月5日のインストール数

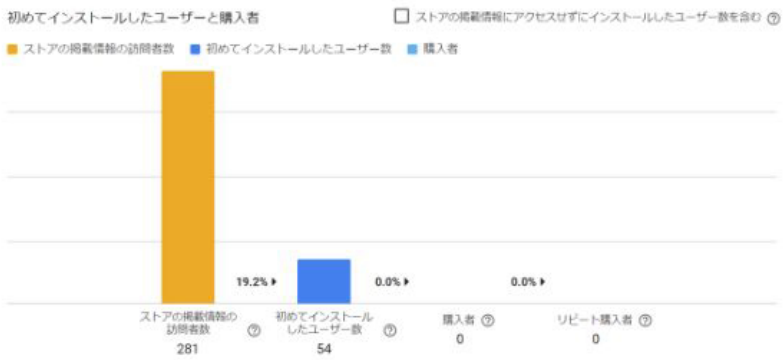



図12.5: 5月6日～5月12日のインストール数

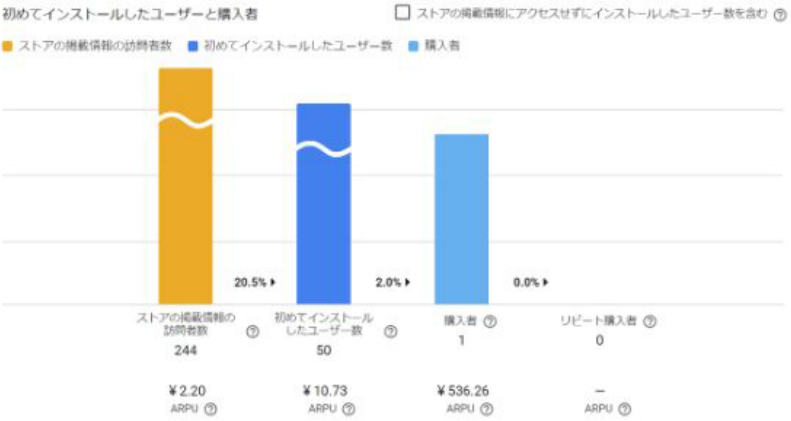



図12.6: 5月13日～5月19日のインストール数

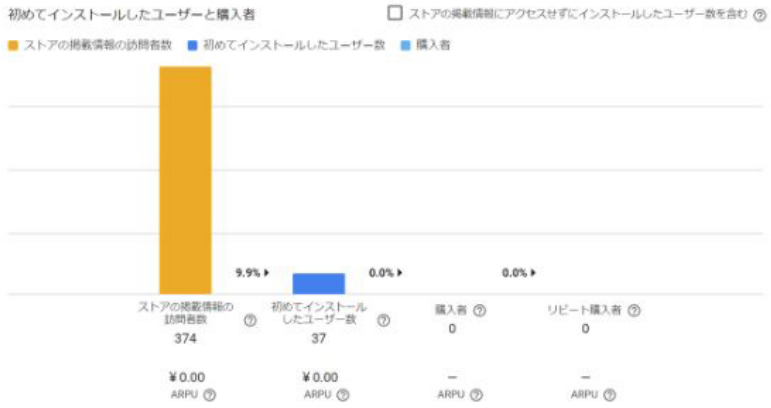
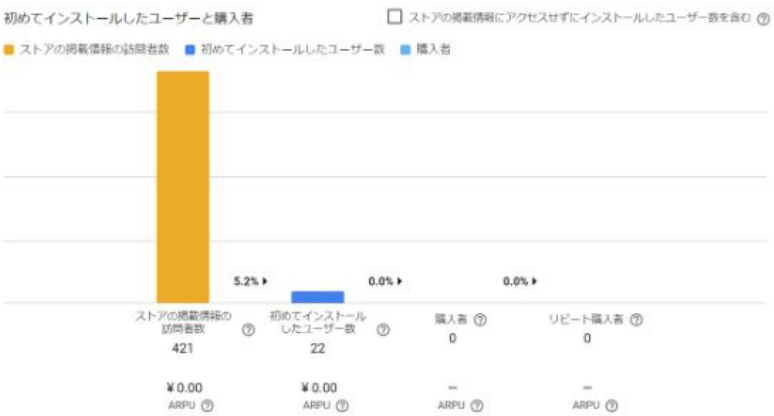



図12.7: 5月20日～5月22日のインストール数




5月13日（3枚目）の画像からストアの訪問者数が増えていることがわかります。4枚目は5月20日～22日で3日間のデータなのでかなり増えています。

ただ、ダウンロード数が伸びていません。これはストアの掲載内容に問題があるのかもしれないですね。

## 12.4　まとめ

と言うことで、憧れのレビューサイトに掲載された振り返りになります！

・ボルダリングで使える遅延再生カメラアプリ「遅カメ」を開発した。

・Applivで紹介されたが、ダウンロード数は変わらなかった。

・ただし、プレイストアへの訪問者数は増えている。

・ストアの掲載内容次第ではダウンロードが伸びる可能性もある。

アプリ開発を始めた頃からすれば、1歩ずつ成長しているのかなと思っています。[3]

```
@kohshin1977


本業での鬱憤を個人開発で晴らす開発者。自分が必要なアプリを作っています。

作った後ほったらかしになるのと進捗が遅いのが玉に瑕。

本業はWindowsで映像関連のアプリケーションを作っています。


https://play.google.com/store/apps/developer?id=onsight
```

1. https://play.google.com/store/apps/details?id=com.kohshin.delaycamera
2. https://android.app-liv.jp/005109025/
3. 本稿は執筆者（@kohshin1977）のブログ記事『遅カメVer1.00をリリースしました！』『「遅カメ」がApplivで紹介されました』『Applivで紹介されたらダウンロード数は、、、』を加筆・修正して寄稿したものとなります。

# 第13章　会社が潰れて1ヶ月でリリース＆起業！エンジニア社長の奮闘記

　1人エンジニア社長のイシベ @isbtty7 です。ショート音声SNS「PitPa」を1ヶ月で開発して、会社を立ち上げました。

　なぜそんなことをやろうと思ったのか。どのようにやったのか。何を大切にしたのか。PitPaの創業エピソードをお伝えします。「つい個人開発で起業したくなる」と思ってもらえたら最高です。[1]

図13.1: PitPaの画面

## 13.1　はじめに: ある日のPitPaへの音声投稿

**「株式会社PitPaを登記してきました！」「頑張っていきたいと思います！」**

自作アプリ「PitPa」に声を吹き込んだ。60秒までのショート音声を投稿するSNSだ。まだ自分のデータしか入っていない。

## 13.2　1st Mission - 1ヶ月でアプリをリリースせよ

大企業からベンチャーに

勤めていた大企業を辞めた。きっかけは外の世界を知ったこと。アメリカに滞在して、スタートアップの人々から話を聞いた。ハーバード大学出身の連続起業家は「こんな未来がきっと来るはずよ」とビジョンを掲げていた。胸が踊った。

何かやりたいと思った。あるベンチャー企業で副業をした。仕事を探すのは難しくなかった。もともとiOSの個人開発者として、ゲームやクイズのアプリをいくつか公開していた。大企業で開発リーダーの経験もあった。示せる実績があった。

副業先のすぐ隣が「TikTok」のオフィスだった。TikTokはすごかった。若年層に大ヒットして急成長。隣で見ているだけの自分。このままでいいのか。焦りを覚えた。

このまま結婚して、子供が生まれて。そうなったら自分は挑戦できなくなってしまうのではないか。一度きりの人生。リスクを取るなら、今ではないのか。今やらなかったら一生後悔するのではないか。大企業を飛び出した。

ベンチャーから無職に

副業先のベンチャーに転職した。元締めはアメリカの会社だ。アジア市場に注目しているらしい。グループ会社が中国で音声Q&Aアプリを成功させている。芸能人が声で質問に答えるアプリ。これを日本にローカライズする会社だ。

全てがすごい勢いだった。ガンガン投資する。「こんなにお金を使って大丈夫なのか」と心配になる。案の定、あっという間に倒産した。「お金がなくなったので終わりです」と言われて、はい解散。これがアメリカか。これが中国か。全てがすごい勢いだった。

無職になった。せっかく大企業を飛び出したのに。こんなはずではなかった。周りの人たちは呆れていた。何も言い返せなかった。多くを失った。

無職から起業家に

手元に残っているのは、iOSの開発スキルと、行き場のないモチベーションだけ。このまま引き下がるわけにはいかない。見返したい。プロダクトを作ってヒットさせたい。

そのためには何が必要か。ベンチャーキャピタル（VC）から投資を受けて、グロースにコミットするのが一番の近道だ。ベンチャー企業で資金繰りに携わって学んだことがある。シードラウンドの資金調達においてVCは「市場」と「チーム」を見る。音声市場に可能性があることは多くのVCが同意してくれた。

問題はチームだ。どうやってチームを信頼してもらうか。プレゼンが上手い人なら方法はあるのだろう。口下手な自分には無理だ。ならば実績を示すしかない。

自分にできることは何かと考えた。これまでエンジニアとして数多くの現場でコードを書いてきた。誰よりも第一線でプロダクトを作ってきた。アプリ開発なら負けない自信がある。だから「1ヶ月でアプリを作った」という実績を示そう。自分には開発スキルしか残っていない。これをぶつけよう。

## コーディング、コーディング、コーディング

1ヶ月、何も考えずに、ただひたすらコードを書いた。ユーザーに対しても、VCに対しても、自分自身に対しても、それなりのクオリティのプロダクトを見せたかった。だから開発スコープは徹底的に絞った。余計な機能はつけない。ログインと音声投稿と音声再生だけ。コンテンツの更新はできない。削除もできない。必要なら後で開発すればいい。何かあったら手動でDBを叩けばいい。

音声は60秒に制限した。すでに長時間音声のサービスはある。ラジオや動画に近い体験だ。だから逆を狙った。Twitterの140文字制限と同じだ。短い時間に絞るからこそ、投稿しやすくなる・再生しやすくなる。未開拓のユーザーに届けることができる。30分間スマホに1人で話しかけるのは辛い。60秒なら現実的だ。また、エンジニアとしては短時間音声のほうが開発しやすい。長くて重い音声ファイルを処理しなくて済むからだ。

技術要素は「勉強しなくていいやつ」を選んだ。過去に仕事で使った技術だけ。メインはiOSアプリだ。学生時代にアルバイトでゲームアプリを作ってから、ずっとiOSが専門だった。AndroidやWEBは初期リリース対象外にした。最低限のWebAPIはFlaskというPythonのフレームワークにした。インフラはGoogle Apps Engine（GAE）を使った。データベースはCloud SQL。ストレージはGoogle Cloud Strage（GCS）。FlaskとGCPは前職で使ったことがある。いざとなれば前職の同僚に質問できる。

朝起きて、コードを書いて、夜に寝る。飯はしっかり食べる。これまでエンジニアとして最前線で働いてきたのだ。自分にとっては一番楽な時間だった。

## 起業家の第一歩: Demo Days

サービスが形になった後のほうが辛かった。本当に売れるのか。考える時間ができた。不安に押しつぶされそうだ。VCに見せて否定されたら、きっと心が持たない。

心配は杞憂だった。肯定的な反応が多かった。尊敬する投資家から「1ヶ月でこのクオリティが作れるなら一緒にやろう」「成功すると思うからサポートしたい」と言ってもらえた。気持ちが楽になった。第三者に認めてもらえたのは嬉しかった。

VCとの壁打ちをしばらく続けた。オープンオフィスといって、申し込めば誰でも無料で相談できる。ただし20分だけ。何を相談するか準備をして臨んだ。PitPaの60秒制限と同じだ。時間が短いからこそ、論点がシャープになる。有意義な時間を過ごせた。

投資家によって投資基準や助言のポイントは違う。多様な観点でブラッシュアップできた。だが全ての意見を採用する余裕はない。割り切りながら壁打ちを続けた。

## プロダクトを作れる人材は強い

どの投資家と組むか。できれば経験豊富なエンジェル投資家と組みたいと考えた。自分で事業を立ち上げて財を築いた起業家たち。彼らのノウハウがあれば、一気にサービスをグロースできるはずだ。エンジェルは「あそこが投資するなら自分も投資するよ」と言う人が多かった。なので彼らの信頼を得ているVCと組むことにした。

繰り返しになるが、狙った投資家と組むには、信頼を得る必要がある。そのために「1ヶ月でアプリを作った」という実績を示した。自分がエンジニアだからできたことだ。プロのエンジニアが全てを投げ捨てたら、1ヶ月でサービスを作り込める。

反対にエンジニアがいないスタートアップは大変だと思う。企画者の意図がエンジニアに100%伝わるわけではない。CTOを募集しても簡単には見つからないだろう。実力のあるエンジニアは選べる立場にある。「技術が分からないやつと一緒に働くのは嫌だ」と言うだろう。自分で考えて自分でプロダクトを作るスキルは、いまの時代において強いと思う。まさに個人開発によって身につくスキルだ。

## 13.3　2nd Mission - 10万再生を目指せ

図13.2: PitPaの使い方


2018年8月12日にプロジェクトを始めた。1人でアプリを開発して、会社を作って、9月10日にリリースした。1ヶ月でリリースできた。

PitPaのユーザー獲得・紹介 - プレスリリースと口コミ

SNSのフォロワーに使ってもらうところから始めた。執筆時点現在（リリース2ヶ月）で1日あたり数百インストールのペースで伸びている。流入したユーザーの定着率は約20%。5人に1人は使い続けている。無料のToCサービスとしては極めて高い数字だ。

PitPaはユーザーの声で出来ている。「声を投稿する」「人の声を聞く」という体験は新鮮だ。ユーザーの数だけ声がある。声の数だけ体験がある。だから面白い。だからユーザーが定着する。声の魅力に気付いてしまったら、テキストでは満足できなくなる。

「音声版のTwitterを目指します」とPR Timesに送った。音声の体験に少しでも興味のあるユーザーを獲得するためだ。プレスリリースはNewsPicksに掲載された。頭の良い人たち、堀江貴文氏などが「これは上手くいかないでしょ」と批判して、そのコメントを見た人が使い始めてくれた。分かりやすいキャッチフレーズにしたことで、（批判とは言えど）コメントを引き出すことができたと思っている。

しばらくして有名ブロガーの紹介で利用者が急増した。テキストを極め尽くしたブロガーは、次のメディアを求めているらしい。ブロガーの利用は想像していなかった。ToCサービスが面白いのはこれだ。ユーザーが新しい使い方を発見して、その面白さを伝えてくれる。それを見た他のユーザーがPitPaを使い始める。そしてまた新しい可能性が広がる。PitPaはユーザーの声で出来ている。やってみないとわからないことだらけだ。

PitPaのシステム保守・運用 - パフォーマンスチューニング

サービス成長に伴い、パフォーマンスが顕著に悪化した。音声投稿時にフォロワーにお知らせプッシュを送るのだが、そこを直列処理・同期処理でプログラミングしていた。ユーザー数が増えると、処理時間がその分伸びてしまったのだ。プロのエンジニアなら並列処理・非同期処理でプログラミングするだろう。

並列・非同期で書くべきか。設計やテストが複雑になってしまう。開発に専念できる大企業の仕事ならそれでもいい。PitPaは1人体制だ。発想を切り替えたい。

そもそも今の仕様はクールではない。一般的にプッシュ通知は「アプリを開くきっかけ」として有効なリテンション施策だ。しかし、投稿のたびにフォロワー全員にプッシュを送るのは、さすがにしつこい。プッシュ通知は送りたい。だけど体験を損ねたくない。

そこで、送るタイミングだけを切り替えることにした。いいね！されたとき、フォローされたとき、投稿者本人にだけプッシュ配信する。1アクションにつき、最大で1人に対しての配信だ。フォロワーへのお知らせだと、100人フォロワーがいたら配信数は100だ。これで配信数は大幅に減らせる。処理の書き方は変えていないので、高度なプログラミングやシステムテストに時間を費やすこともない。

むしろプッシュ通知が「投稿を誰かに見てもらえた」という喜びを生み出す機会になる。ユーザー体験を向上させながら、システムのパフォーマンスも改善する。1人エンジニア体制で簡単に実現できる。技術的負債も増えない。バグも生じにくい。自分自身がエンジニアであり、かつ意思決定者だからこそ打てた一手だった。

システム保守・運用は大事だ。守りなしにサービスは成長できない。だけど、教科書通りにやるだけが答えじゃない。1人で開発するからこそ、できる発想もある。

## PitPaの機能追加・修正① - 連続再生機能

プロダクトのことはユーザーの声を聞け。この考え方を大切にしている。機能追加開発についても、要望の多いものを優先している。

最初に追加したのは「連続再生」機能。それまではひとつずつ音声を選択して聴く必要があった。音声コンテンツは最大60秒。60秒ごとに次の再生ボタンをタップしないといけない。面倒臭い。そんな意見が多かった。だから連続再生機能。

これで歩きながらでも聴けるようになる。リリースして真っ先に自分で使ってみた。ユーザーの音声を聴きながらオフィスに出勤した。最高の体験だった。

ひとつひとつの音声には60秒制限がある。メッセージがシャープに伝わる。なによりも声だと話し手の感情が伝わる。温かみを感じた。朝に元気をもらえた。今日も頑張ろうと思えた。連続再生を作ってよかった。そう思った。

## PitPaの機能追加・修正② - フォロワーリストと音声コメント

①小さな案件だがフォロワー一覧画面の改修がある。もともと古い順にフォロワーが並んでいた。一番上に表示されるのが最古のレコード。最新のレコードは一番下。通知が来て「誰にフォローされたのかな」と画面を

開いても、下までスクロールしないと分からない。改修要望が寄せられた。新しくフォローされた順番で表示するように修正した。システムとしては些細な違いだ。この積み重ねで使いやすいアプリになる。

②音声コメント機能も追加した。誰かの声に対して、声で返信できる。自分の投稿に対して、誰かが声で感想をくれる。この機能は好評だった。他のSNSだと、テキストや写真を投稿して「いいね」の数を競い合い、ユーザーは疲れてしまう。PitPaは違う。生の声でコメントが寄せられる。誰かが音声を吹き込んでくれる。生きたコミュニケーションができる。今までにない体験だ。

エンジニアとしてコードを書けることが自分の強みだ。だからこそプロダクトを磨き込んで、ユーザーに最高の体験を提供したい。そう思って週に1回の頻度でリリースし続けている。気付いたときには、最初のリリースから1ヶ月で10万再生を実現できていた。

図13.3: PitPaのグロース

## 13.4　3rd Mission - グロースハックせよ

毎週リリースができている理由は何か。それはPitPaのサービス上でユーザーと「声」で頻繁にコミュニケーションを取っているからだと思う。

PitPaのユーザーヒアリング① - 交流する

アプリ内でインタビューしている。「最近投稿しなくなった方、ぜひ理由を知りたいです。使いにくいところを直します」と音声で相談する。コメント欄で意見をもらう。

作ろうとしている機能について、リリースの状況について、音声で発信する。「こういう機能、どう思いますか」「週末には新機能リリースする予定です」「Appleの審査が長引いています」「リリースしたのでアップデートをお願いします」

PitPaはコミュニケーションのアプリだ。開発者から声を掛けていきたい。

PitPaのユーザーヒアリング② - 使い方を聞く

「iPhoneを目で見ながら音声を聴く。PitPaの音声を垂れ流して画面は見ない。どちらが多いですか？」といった質問もする。シチュエーションによって適切なUIが変わるからだ。見て面白いアプリ。垂れ流しやすいアプリ。どっちにデザインするか。

「音声入力するときは、一時停止を使っていますか？一気に喋りきっていますか？」といった質問もする。投稿画面を改善するためだ。撮り直すなら「一時停止」機能は不要ではないか。画面1つ1つの要素について、役割を考え直すヒントにした。

PitPaのユーザーヒアリング③ - 声で察する

これが声の魅力だ。声の雰囲気でユーザーの本気度が伝わる。求められている機能を早く作らなければ、という気持ちになる。

「実はこの機能を使っていない」という声も聞ける。余計な機能は削除できる。必要なものを必要なリソースで作るためには大事なことだ。

前職だとユーザーとの距離が遠かった。妄想で設計してリリースした。結局ユーザーの求めるものじゃなかったこともある。ユーザーとの今の関係は大事にしたい。

PitPaのグロースハック① - Twitterシェア機能

「投稿する人」と「聴く人」がいる。どこから磨き込むべきか。よくあるタマゴとニワトリの問題だ。最初の時点では「投稿する人」のための施策を優先した。

1つがTwitterへのシェア機能だ。PitPa内に聴く人がいなくても、投稿したくなる。その仕組みを考えた。TikTokを参考にした。TikTokで作った動画を端末に保存し、Twitterに載せる。TikTokを使っていないユーザーに向けて発信する。そういう使い方をしているユーザーが大勢いる。同様にPitPaを音声作成ツールとして提供しよう。

最初のシェア機能はシンプルだった。共有URLを開くとPitPaのWEBサイトで再生できる。ログインしなくても聴ける。一番作りやすい仕様だ。サービス内での再生数も増えるだろう。新規ユーザーの流入も期待できる。

次にTwitter単体で聴けるようにした。より多くの人に音声を聴いてもらえるはずだ。Twitterに音声再生機能はないので、動画ファイルに変換してアップロードした。上手くいった。圧倒的に再生数が伸びた。「Twitterシェアだと再生数が伸びやすいですよ」「ぜひTwitterシェアを使ってください」そう呼びかけた。

PitPaのグロースハック② - 端末保存機能

同じタイミングでもう1つ機能を追加した。投稿した音声ファイルをユーザーの端末に保存する機能だ。システム不具合でTwitterシェアが失敗しても自分で投稿できる。ブログやInstagramなど他のSNSプラットフォームにシェアできる。

リンク共有で新規ユーザーにサイトを訪問してもらう。それが運営者としては一番望ましい。だけどそれではダメだ。最高のユーザー体験を提供するためにあえて囲い込まない。「また投稿しよう」「PitPaを使い続けよう」と思ってもらうことが最優先だ。でないと新規サービスはユーザーに選んでもらえない。ビジネスとして広がらない。

たった1人のエンジニアが作った、できたばかりのプロダクト。そんな1つのサービスで全てを完結させてユーザーを囲い込もうとするなんて無理だ。ついそんな構想を描きたくなる。だけどそれはユーザーの声を無視するということだ。「個人開発」を「独りよがりの開発」と履き違えているだけだ。グロースハックは、ユーザーの声を聴いて、ユーザーに声を掛けることだと思う。

## 13.5　4th Mission - イノベーションを起こせ

PitPaの将来展開 - 音声の民主化

音声には可能性がある。10年後には声の投稿が当たり前の時代が来ると思う。

今のプラットフォームとしてはTwitterやInstagramが強い。TwitterやInstagramには画像形式で「言葉」を投稿するユーザーが大勢いる。あれはメッセージを伝えたいのではないか。だとしたら音声のほうがメッセージが伝わるのではないか。

音声の良さは他にもある。視覚を使わないことだ。ある芸能人の投稿動画を見ると、カンペを読むような目の動きが気になった。動画は情報量が多い。だからこそ投稿者の負荷が大きい。いずれ動画疲れの芸能人が音声市場に流れ込むだろう。

現時点だとSpotifyのように音楽を垂れ流すのがメインだ。しかしPitPaをリリースしてから、誰もが音声を投稿できる「音声の民主化」を想像するようになった。音声は魅力的だ。だから高いユーザー定着率を実現できている。いずれ「音声広告」のような新しいビジネスが生まれるはずだ。

音声投稿の障壁

音声には難しさもある。例えば、電車内だと使いにくい。家に帰ったら音声投稿しよう。いざ帰宅してPitPaを開く。何を話そうとしていたのか忘れている。

多くのユーザーから要望が寄せられて、下書き機能をリリースした。せめてタイトルだけでもテキストで書いておこう、という機能だ。「そうだった！あれを話そうと思っていたんだ！」と思い出せる。

また、テキスト投稿に比べて心理的なハードルがあるのは事実だ。削除機能は早い段階で追加した。いつでも消せるという安心感。多少は投稿しやすくなったはずだ。

InstagramやSnapchatで動画が1日で消える仕組みは上手いと思う。勢いで投稿できる。恥のかき捨て。だからこそ人間味のある面白いコンテンツになる。見ていて面白い。感情が乗る。話題になる。

これが音声だったらどうだろう。どんな面白いコンテンツが生まれるだろう。どんな素敵な体験をユーザーに提供できるだろう。同じ仕組みを作ったほうがいいだろうか。

検討中の施策

①サービス成長に伴って新しいニーズが出てきた。増えた投稿をまとめるアルバム機能。Spotifyで音楽をまとめて聴くように、テーマで音声をまとめる体験だ。

②ユーザー数が増えたので、全体に向けて質問・お題を投げかける機能も作ろうと思う。好きな企画に参加して音声を投稿する。1人に対する質問はもともと可能だった。そのコンセプトを拡張した機能だ。

③ミートアップイベント（いわゆるオフ会）も開催する。対面ならではの本音を聴けるチャンスだ。ユーザーと交流して、改善のヒントを得たい。

④SEOは課題だ。音声ファイルだと検索エンジンに乗らない。そこで音声のテキスト変換技術に注目している。これなら音声サービスでSEOを実現できるのではないか。

⑤Androidアプリも作る。自分はAndroid開発が得意ではないので、友人に依頼した。国内でAndroidユーザーの比率は上がっているらしい。それに将来的に海外を狙うならAndroidアプリは必要になる。

アメリカや中国の勢いを知っている。たった一度きりの人生で、リスクを取るなら、今ではないのか。今やらなかったら一生後悔するのではないか。そう思って大企業を飛び出した。だからこそ海外は狙いたい。プロダクトを磨き、いずれ世界を目指したい。

とはいえ、まずは目先。正しいものを正しく作る。ユーザーの声を大切にしながら。[2]

## 13.6　おわりに: ある日のPitPaへの音声投稿

**PitPaの開発者のイシベです。今日は感謝をまとめようかなと思います。**

**先日ですね、あの、たくさんのユーザーさんにアプリをダウンロードしていただいてですね。えー、たくさん使っていただいて、えー、フィードバックをいただきました。そちら、とても感謝しています。**

**あの、声で、人と繋がることの価値っていうのは、実際あるんだなと、いうところを体感することができました。その価値を感じている方々にですね、「最っっっ高のサービス」を届けたいと思います**

**いただいたフィードバック、全て、改修していきたいと思うんですけど、一気に全てを片付けることはできないので、優先度をつけながら、丁寧に一個一個リリースしていきたいなと思っています。リリースした際には、改めてですね、この「声」を使って、発信していこうかなと思っています。**

**えー、以上です。**[3]

```
イシベタツヤ / @isbtty7

株式会社PitPa代表取締役CEO。グロースハックが得意なiOSエンジニア。
学生時代はアルバイト先でゲームアプリを量産。会社員時代は個人でクイズアプリや
割り勘アプリを量産。独立後はVoiceTechに可能性を見出して、開発着手から1ヶ月で
ショート音声SNS「PitPa」をリリース。

https://pitpa.jp
```

1. 本稿はイシベタツヤ（監修）との居酒屋談義を@yuzutas0が代理執筆したものとなります。
2. 編集注:執筆〜出版の間に実現している機能もあります。
3. @isbtty のボイス：たくさんFBいただけて嬉しいです / PitPa -「音声」でつぶやくSNS https://pitpa.jp/share/timeline/smXbZxhxgBDverYxhP

# 第14章　個人開発のモチベーションが続かない、作り終わらない。その原因と対策

　株式会社クリモの@nabettuと申します。「ためしがき」というWebサービスを運営しています。個人でサービスを作る時に起きやすいモチベーションが続かない。いつの間にか開発をすることが辛くなってくる。そういった「**個人開発あるある**」に陥る原因と対策についてまとめました。

## 14.1　クリエイターの創作活動を応援するサイト「ためしがき」

「ためしがき」[1]では、好きな文章や文字を入力することで、色々な種類のフリーフォントを試せます。ねとらぼ、窓の杜、Gigazine、ITmediaといったメディアに取り上げていただきました。

図14.1: ためしがきサイトページ


執筆（2019年8月）時点で扱っているフォントは95種類です。クリエイターが自力で95種類のフォントを試すのは容易ではありません。色々なサイトを巡り、ファイルをダウンロードして、手元で文字を当てはめる。手間がかかる作業です。かといって、用意されたサンプルテキストだけを参考にしても、いざ文字を当てはめてみたら、期待した出来栄えにならないかもしれません。

このWEBサービスはクリエイターの「面倒臭い」や「何か違う」を解消します。創作活動がもっと楽しくなるかなと思って作りました。個人開発をするときにはぜひご活用ください。

技術面ではNuxt.js（Vue.jsのフレームワーク）とFirebaseの組み合わせで作っています。コンテンツ管理はContentful（ヘッドレスCMS）を使い、CircleCIでCMSの内容とソースをまとめてFirebase Hostingにデプロイしています。

## 14.2　クリエイターの創作活動を挫く「どうしてこうなった」の数々

　このように、個人開発は「**自分の作りたいもの**」を「**自分の使いたい技術**」で「**自分の好き**」に作れる**最高の舞台**です。クリエイターならば、やらないという選択肢はないでしょう。しかし、何も考えずに足を踏み入れると、思ったよりも険しい道だということに気が付くことでしょう。

　自分が作りたくて作っていたはずなのに、いつの間にか失ってしまうモチベーション。何ヶ月経っても作り終わらない、まるで現代のサグラダ・ファミリア。仕事が忙しくなって一週間放置しただけで「え、なにこのクソコード」と毒づいてしまうほど荒れ果てたmasterブランチ。あのころ目指していた「一発当てて平日昼間から温かいこたつでレディボーデンを頬張っている自分の姿」はどこへやら。

　そんな目も当てられない状況、私にもありました。「**どうしてこうなった**」と頭を抱え開発を諦めてしまわないよう、私なりに原因と対策をまとめてみました。これから個人開発をしていきたいと考えている人達のご参考になればと思います。[2]

## 14.3　原因1: 構想がでかすぎて作りきれない

まずはこれ。「すっげーいいアイディア思いついた！！！」とテンションが上がり「どうせなら高いレンタルサーバーを借りよう」などと（そもそも借りるのは公開直前でいい）言い出すのですが、気づいていない落とし穴があります。スケールがデカイのはいいことですが、本当にそれは**自分ひとりで作りきれるサイズ**でしょうか。まず始める前に、なんとなく全体像を整理しましょう。例えばTwitterのような、タイムラインが一つのサービスを作ると考えて、サービスに必要な機能を考えます。

・トップページ
・ユーザー登録ページ
・ユーザー登録機能
・ユーザーページ
・ユーザー情報の編集
・ユーザー退会機能
・投稿一覧ページ
・投稿詳細ページ
・投稿機能
・投稿削除機能

などが機能として必要になりそうだなと整理します。それを元に、本当に作りきれるのかを考えます。

1ヶ月で作りきれるものにしよう

なんとなくでいいので工数を計算しましょう。**空いてる時間をすべてつぎ込んでも1ヶ月で作れないなら、サービスのサイズを考え直したほうがいい**です。確実にガウディります。

削るところは削ってまずはMVPを出す

作りきれる人は考えなくていいのですが、そうではない人は**１ヶ月で作りきることができるMVP**を考えましょう。最小限の機能を供えた製品（MVP：Minimum Viable Product）を作り、顧客の反応を見ながら改善するリーンスタートアップの手法です。

先程考えたサービスに置き換えて考えると、ログインもマイページもフォロー機能も一旦置いておきます。ログイン機能を作るのに1週間。マイページを表示する機能を追加するのにのに1週間。自己紹介文を編集できる機能を追加するのに1週間。ユーザー画像が変更できるように画像アップロード機能を追加するのに1週

間。毎日がんばって1ヶ月が経過しても、まだ他のユーザーの投稿さえ見えません。そんなことをやってるうちに**モチベーションはどっかに行ってしまいます**。

まずは、投稿画面と一覧画面だけを作り、どんな感じのものになるかを見てみましょう。ひとまずそのサービスで一番コアになる部分を分かるように作りましょう！

公開できる部分に切って、そこだけ作ってユーザーテストする

そんな流れでMVPを作ったら、知り合いに見せてみましょう。TOPページの説明はまだ作っていないので、自分で簡単に使い方を説明してから、投稿画面を触ってもらいましょう。「よくわかんないね。hahaha」とか「これってTwitterと何が違うの？」と、言われて落ち込むかもしれません。反対に「あ〜これ、こういう使い方もできるね」「逆に『にゃーん』しか呟けないようにしたら？」など嬉しい意見がもらえるかもしれません。[3]モチベーションが逃げる前に、少しずつでいいのでプラスの声を聞いて、歩みを止めないようにしましょう。

作る時間をしっかりとれるように計画しよう

中だるみを防ぐために「一人開発合宿！」と叫びながら「ビジネスホテルにPCだけ持って一泊する」など試してみるのもいいです。とにかく時間を確保しないことには進捗は生まれません。**上司に媚びを売るための飲み会なんか行ってる場合じゃない**ですよ！

## 14.4　原因2: 技術的にしんどい

　次はこれ。作る機能を絞り、時間を確保して、いざ作り始めてみた。そうしたら思ったよりも難しい。「ProgateとRailsチュートリアルを終わらせた俺に作れないものなど無かったはずなのに……」といった心境になり、心が折れそうになってしまうかもしれません。

使ったことがない技術はできるだけ使わない。使っても一つだけにしよう

　**未経験の技術でやろうとするのは個人開発アンチパターン**です。よっぽど作りたいものでなければ作りきれません。ここは的を絞り、ちょっと背伸びすれば作れるレベルのものにして、壮大な構想は後にとっておきましょう。千里の道も一歩からです。

「新しい技術の習得」と「新規開発」はできるだけ分離しよう

　自分の作れる範囲のものなんて作っても仕方がない。今のスキルでは大したものは作れない。そう考えるようでしたら**技術の習得と新規開発はできるだけ分離**しましょう。

　新しい技術の習得中は、サービス開発が進んでないように見えて、心が病んでしまいます。この悲劇を防ぎましょう。サービス開発のToDoリストとは別に、技術の習得チェックリストを作るのがおすすめです。出来るようになった事や学んだ事を記録して、ステップアップが見えるようにしておくといいと思います。

　また、「技術の習得のために作っている」というパターンならあまり心配はないです。その場合は作れなくとも技術を習得できれば目標は達成ですから。

汚いコードでも、まずは動くものを作ろう

　「完璧を目指すよりまず終わらせろ」（Done is better than perfect）ってやつですね。ひとまず動くものを書きましょう。ある程度動くようになったことを確認してから、徐々にリファクタリングをしていきましょう。コンポーネント指向で作る場合も同じです。始めは一つのコンポーネント（ページ）に全部の要素を詰めこみ、余裕ができたら徐々に分けていけば大丈夫です。

動的な部分も静的なもので一旦作る

　**後から入る動的な数字も、まずは決め打ちで書いて**後から直しましょう。細部にこだわるより一旦全体を通して使えるものにするのが先です！シェア数やPVなどの数字があとから入る部分は、ひとまず全部「999」でベタ書きしてはいかがでしょうか。ユーザーアイコンはpravatarなどのサービスを利用して、一旦仮の画像で埋めておきましょう。[4]

### 最初はiOSだけに対応する

レスポンシブ対応・他OS対応は先送りでいいです。公開してから徐々に直して行けばいいのです。このご時世、スマホで見れるものを作れば、PC版の対応は後からで十分です！

PCではscaleなどでスマホの画面を2倍に伸ばす。余白はそれっぽい背景色を引いて終わりにする。そのレベルで十分です。スマホメインのサイトの場合はこれで運用しているサイトも意外とあります。

### CSSを一切書かないでコアな部分だけ先に作る

「CSS書くの辛いマン」という人、結構いると思います。一旦はbootstrapなどのCSSフレームワークを入れただけにしておいて、後から書き足しましょう。今ではBulma.io[5]などのbootstrap以外のCSSフレームワークもたくさんあり、色味をちょっと変えるだけで十分リッチになります。自分で書くのは後回しにしましょう。また、Semantic UI[6]などSPAライブラリー向けにコンポーネントフレームワークと呼ばれるものもあります。それらを利用してみてもいいかもしれません。

### 外注する

絶対作ってやるぞ！という場合、もう自分が苦手な部分は外注してみるのも 1 つの方法です。例えばクイズを誰でも作れるサービス[7]のような CGM（Consumer Generated Media）を作る場合は、コンテンツの外注を試してみましょう。10問でも誰かが作ったデータが入っていればそれっぽい感じになります。クラウドソーシング[8]を利用して、コンテンツの制作を手伝ってもらいましょう。

## 14.5　原因3: 作ってる間に似たようなサービスが先に出てきた

　これもあるあるですね。個人だろうが仕事だろうが、こういったことはよくあります。Webサービスなんてだいたい「掲示板」の延長ですから。「カキコ」したら一緒なんですよ。

同じ事を考える人はたくさんいるので、気にしない

　「Googleは世界初の検索エンジンではない」－－ 今日はこれだけ覚えて帰ってください。はっきり言って世界中探せば、自分が知らないだけで似たようなサービスはたくさんあります。気にしても仕方がないのです。

　そして、違う人が似たようなサービスを作ったとしても、**全く同じサービスにはならない**です。**機能は似ていても、思想まで一致することはない**と思います。FacebookもTwitterも、結局は何かしら書き込んでタイムラインに表示するだけなのに、こんなにも違います。思想が違うと方向性も変わるので、自分の考えをしっかりと固める機会にすればいいのではないでしょうか。

## 14.6　原因4: 他のメンバーが動かないor自分の負担が大きい

　個人ではなくチームで開発する場合の話です。人によって、モチベーションもかけられる時間も、コストもリスクも変わってしまいます。メンバーを募集すると、集まるときは意外とすぐに集まるものなのです。しかし、一緒に開発をする以上、メンバーをある程度は選定しないと、スキルレベルのばらつきがどうしても生じてしまいます。

割り切って全部自分でやる

　動かないメンバーは無視して、自分ですべてやってしまいましょう。下手に頼るとコミュニケーションコストが高くなり、意思決定が遅くなります。その時の注意点ですが、権利関係でもめないように、納得の行く形でエビデンスを残しておきましょう。口を出すだけのメンバーに「アイデアを出したから俺にも取り分あるだろ」などと言われないように、事前にきっちり決めることは決めておいてください。

非情になる

　なってください（笑）。仕事じゃないプロジェクトでわいわい集まったとしても、本当に仲の良いメンバーかレベルの高いメンバーが揃わなければ上手くいきません。プロジェクトを進めたい気持ちがあるのならば、なぁなぁで事を進めないことが大切です。

　そして、「**プロジェクトにとってプラスになることしかしない**」と割り切ってください。例えば「ファミレスで会議したい」とメンバーが言い出したとしましょう。それで進捗が出るならいいですが、決めることや議論したいことが固まっていないのに会議しても何の意味もありません。

　オンラインのやり取りだけでもWebサービスは開発できます。無駄をなくしましょう。意思決定をする上で大切なことや、プロジェクトを完成に近づける事だけに力を注いでください。

一旦解散する

　煩わしいなと思ったら、一度解散してしまうのも手です。またなにか思いついたときに集まればいいじゃないですか。もし「解散するぞ」となった時に「自分は続けたい」というメンバーがいたら、そのときにある資産を誰が持ち帰るのか、という点だけは、もめないようにしましょう。

## 14.7　原因5: 作ったけど誰にも使われなくて悲しい

今度は作りきった後の話です。色々頑張って完成したものの、サービスが使われないのは悲しいですね。でも「一つのサービスを作り切る」ってすごいことです。思い入れがあるなら今までの努力を無駄にしないように、もうちょっとだけ頑張ってみましょう。

### 自分でとにかく使う

自分が「いいな」と思ったから作ったんですよね！ 自分で使ってみて、少しでもサービスのいい所があったら、また友達に見せてみるのもいいでしょう。サービスを色々と使っていくうちに「あ、こんな使いかたできるかも」とか「横展開してこういうのできそう」とか思いつくこともあります。今ではTwitterトレンドによく出てくる『アプリ☆メーカー』[9]も、流行るまでの間は何ヶ月もひたすら自分たちでコンテンツを作り続けていたらしいです。

### 自分が欲しい部分をとにかく作る

サービスを使いまくっていると、今度は「コレも欲しい」「あれも欲しい」となってくるはずです。自分がファーストユーザーになれないサービスなんて、作っても全然面白くありません。自分のための機能をどんどん作っちゃいましょう。自分のためだけの最高のサービスになったとしても、それはそれでいいじゃないですか。

### 他の開発者と交流する

TwitterでWebサービスを作ろうとしている人と交流するのもいいかもしれません。「**#駆け出しエンジニアと繋がりたい**」というTwitterハッシュタグを使っている人もたくさんいます。同じようにWebサービスを作りたい人を探して、お互いに意見を出し合うのもいいかもしれないですね。

また、個人レベルでWebサービスを作っている、または作ろうとしている人達のコミュニティなどもあります。そのコミュニティに参加して「他の個人サービス運営はどうやってるのかなー」と覗いてみたり、意見をもらってもいいかもしれません。**この本の筆者達はみんな個人開発をやっているので相談にのります。**Twitterで気軽に話しかけてみてください！

## 14.8　まとめ

　個人サービスのモチベーション低下問題を未然に防ぐ方法、そして乗り越える方法を私なりに書きました。

　個人で作ることのできるサービスの範囲が年々大きくなっていくWeb業界ではありますが、華々しくデビューを飾るサービスの裏ではたくさんの折れた心達があります。「Webサービスを作ってみたい！」という人は挑戦する前に少し心がけてもらい、心を折らずに作りきって欲しいと思います！

　この話を読んだクリエイターが何とかモチベーションを保ち、面白いWebサービスを作りきって世に出す助けになれば幸いです。

```
渡邊達明 / @nabettu

仙台高専→富士通株式会社→面白法人カヤック→ブロガーの妻と二人で株式会社クリモを
設立。WEBフロントエンド中心の受託開発や保育園問題解決のためのメディアを運営している。
「三度の飯よりものづくり」と言っていたらBMIが17になり健康診断で毎回ひっかかる。
日本語のフリーフォントを一度にためせる「ためしがき」を開発。

https://tameshigaki.jp/
```

1. ためしがき https://tameshigaki.jp/
2. 本稿は執筆者（@nabettu）によるブログ記事『日本語のフリーフォントをまとめて試せるサイト「#ためしがき」をなんで作ったのかと技術的な話』『個人開発のモチベーションが続かない、作り終わらない。原因と対策を考えてみた。』を加筆・修正して寄稿したものとなります。
3. Nyaaan https://nyaaan.haramishio.xyz/
4. pravatar http://pravatar.cc/
5. https://bulma.io/
6. https://semantic-ui.com/
7. https://quiz-maker.site/
8. https://www.lancers.jp/
9. http://appli-maker.jp/

# 第15章　学生エンジニアが語る「技術学習 x 個人開発」のススメ

　こんにちは。まっきー（@macinjoke）と申します。今回は僕が作ったタイピングWebアプリ「Study Typer」[1]についてご紹介します。僕は「技術を楽しもう」と思っています。新しい技術をひとつひとつ学びながら、時間を掛けてデプロイに漕ぎ着けました。3部構成で、前編は自己紹介、中編はStudy Typerの説明、後編は技術面の話をします。

## 15.1　自己紹介

　情報系専攻の修士1年です。初めてのプログラミングは中学生の時で、ボンバーマンのようなゲームを作りました。プログラミングの楽しさを知り、今では趣味やアルバイトでコードをバリバリ書いています。

　最初はWebのバックエンド開発が中心でしたが、Reactに出会ったことがきっかけでフロントエンドにのめり込みました。今ではフロントエンドで食っていくと心に決めています。

## 15.2　Reactと恋に落ちた理由

jQueryのコードを扱ったとき「フロントエンドはカオスになりそうだから関わりたくない」という感想を抱きました。しかし、Reactに出会い、見方がガラっと変わりました。フロントエンドでも安心してライフサイクル・状態管理ができると気付き、心踊らせたのです。

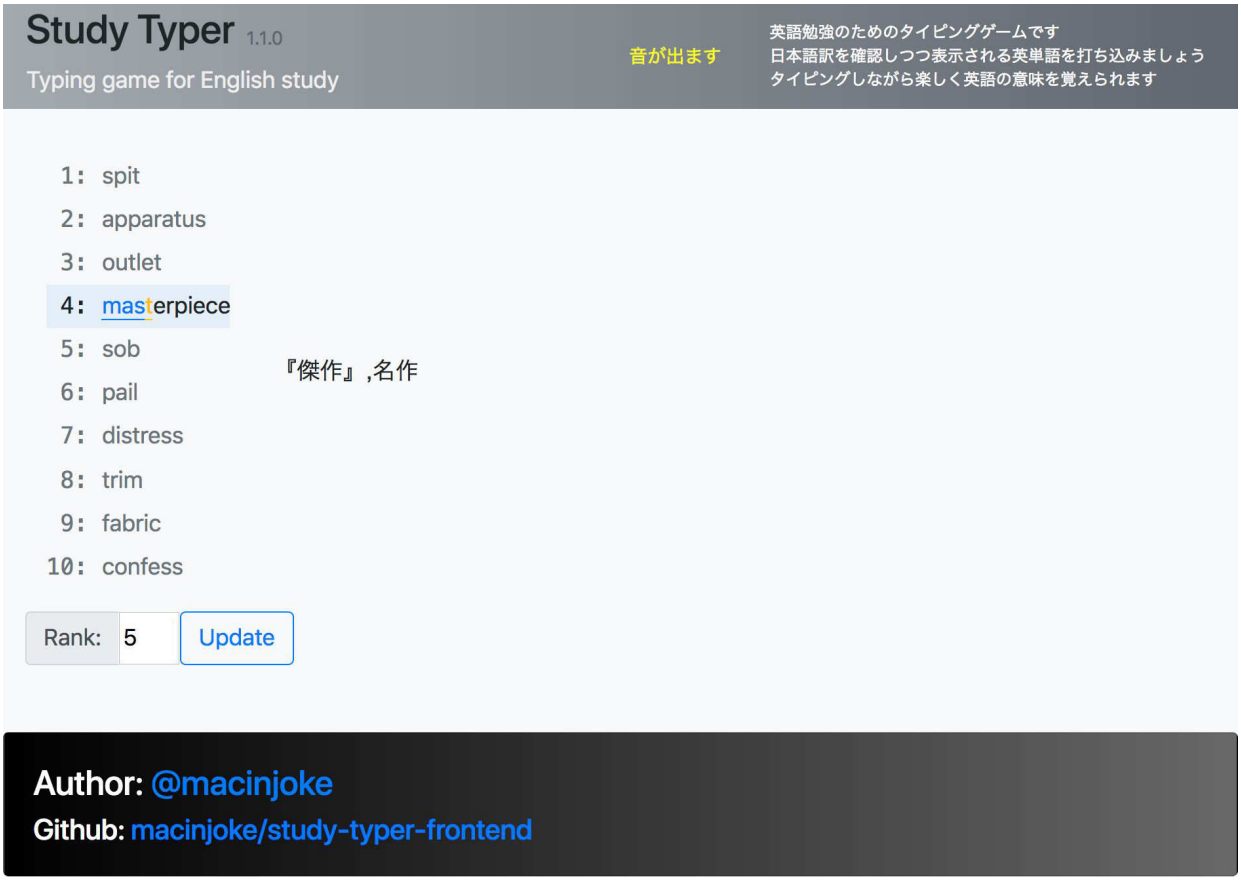
Reactによるコンポーネント指向の開発では、機能と見た目をひとつのコンポーネントに閉じ込めるので、CSSで予期せぬ副作用が出ません。僕はUIデザインにあまり興味がないので、CSSを書くのはデザイナーさんに頑張って欲しいと考えています。コンポーネントごとにデザインを実装できると、分担作業を進めやすいので良い開発体験になりそうです。そのためにReactではStorybookというツールが用意されています。

このようなことが、自分にとっては新鮮で面白いと思い、Reactを極めたいと思うようになりました。

## 15.3　Study Typer とは

本題のStudy Typerについて説明をしていきたいと思います。ずばり「英語学習者のためのWebタイピングゲーム」[2]です。

図15.1: Study Typerのプレイ画面




1ページのみで構成されたWebアプリです。左側に英単語が、右側に日本語訳が書かれています。

日本語の意味を確認しながらタイピングをすることで、英単語の勉強になります。入力するのは日本語ではなく英単語なので、英語タイピングの練習にもなります。

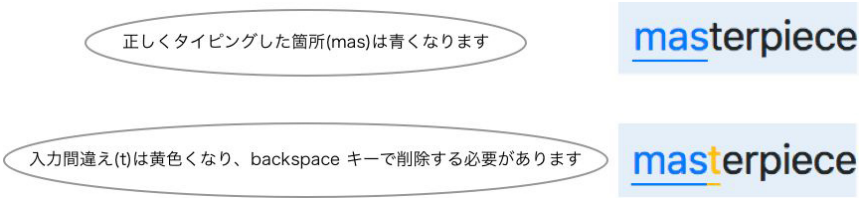
英単語のタイピングの早さというのは、その単語を何回打ってきたかで決まると思います。僕の実体験だと「foreign」という単語を打ち慣れていなかったので何度もタイポしました。

下の方にあるRankパラメータは1〜30まであり、英単語の難易度を変更できます。1が一番簡単で、7まで難易度が上がると難しくなります。1画面で10個の単語をタイピングできるようになっており、10問目が終

わったらまた1問目に戻ります。出題される英単語はランダムです。Updateボタンを押すことで新しい10問の英単語を打てます。

その際に次の英単語の音声が流れます。発音の確認もできて便利です。

図15.2: タイピングの視覚的なフィードバック




想定ユーザーは、高速タイピングでドヤりたい＆英語力UPでモテたいエンジニアです。

なぜ作ることにしたのか

このアプリを作ったのは単にノリです。自分が欲しいと思ったものを技術の勉強がてら作ってみただけです。マネタイズは考えていません。ただ、ノリで作ったからといって、このアプリが無価値だとは思っていません。他にはない便利なアプリなので、ぜひ使ってみてください。

Web上でタイピングゲームを検索すると色々と出てきます。しかし「日本人の英語勉強用」に特化したタイピングゲームは数が絞られます。さらに「英単語」に絞ったらほぼ見つかりません。そして「見た目がクールである」「すぐにゲームが始められる」「**英語の音声が流れる**」といった自分の要望を満たすものはまだ世の中にありませんでした。

今後の改善点

今後以下のような機能追加・改善を考えています。

・UI改善（一気に英単語10個表示している意味がなさそう）
・英単語データの拡充
・ログイン機能: 自分の記録を残す（ログインしなくても使えるように改修するつもりです）
・タイムアタック・スコア機能: 各単語の打ち始めから打ち終わりまでの時間を計測して、日本語の意味をしっかり読みながら競える（学習用タイピングゲームなので）

時間のあるときにゆっくりと作っていきたいと考えています。ご意見や機能提案、バグ報告、プルリクなどWelcomeですのでよろしくお願いします！

## 15.4　技術的なお話 〜どうやって作ったか〜

コードはGitHubで公開しています。

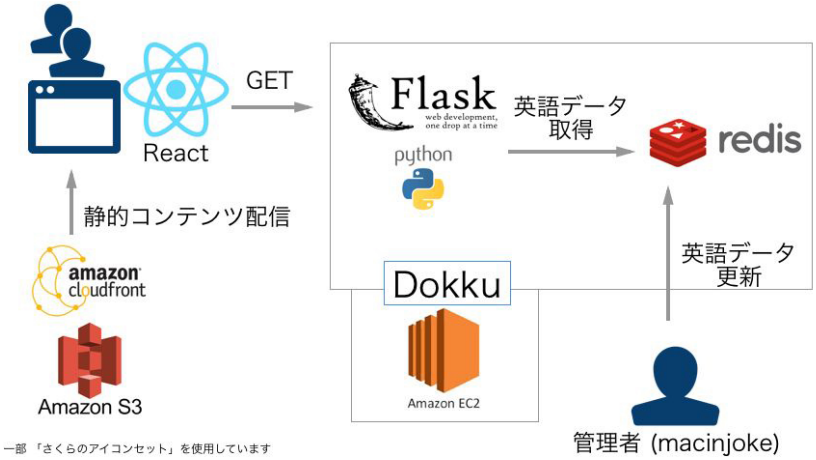
・フロントエンド: https://github.com/macinjoke/study-typer-frontend

・バックエンド: https://github.com/macinjoke/study-typer-backend

## 15.5　システム構成

Study TyperはReactを使ったSPAです。APIにはPythonのFlaskを利用しています。英単語データの保存にはRedisを使いました。システム構成図をお見せします。

図15.3: システム構成図




### Dokku

AWS EC2のUbuntu上に「Dokku」[3]と呼ばれるHerokuライクなPaaSを簡単に立ち上げるツールを入れています。Dokku上のメインアプリとして、Python、Flaskを使ったAPIサーバを立ち上げています。

Dokkuの「Redis Plugin」[4]でRedisを立ち上げ、Flaskアプリと繋げています。アプリ側の REDIS_URL という環境変数に値が入るのでそれを使って接続します。これらは git push コマンドで楽チンデプロイできます。Dokkuを使った構成はコマンドを何回か叩けばポンとすぐできるので簡単でした。

わざわざDokkuを構築するよりも、クラウドPaaSのHerokuを使ったほうが実は簡単です。ですが、Herokuだとまあまあお金を払わないと、Redisのデータを永続化することができません。今回はRedisのデータを永続化したかったので、使い慣れたHerokuは手放しました。

### Redis

Redisにはランクをつけられた英単語とそれに対応する日本語のデータが格納されています。ブラウザからのリクエストによってFlaskサーバがRedisからデータを取得し、レスポンスを返します。データの更新は管理者（僕）しかしません。

なぜ数あるDBの中でRedisを選んだかという話ですが、単純に使ってみたかったからです。とはいえ、後から振り返るとRedisならではの利点がありました。

リレーショナルデータベースや、MongoDBのようなドキュメント型のデータベースと比べるとRedisはレスポンスが早いです。Redisはインメモリのデータベースで、永続化機能を弱める代わりに速度を上げています。今回のアプリではデータの更新を行うのは管理者(僕)だけなので、更新は頻繁に起きず、速度が落ちるような心配もありません。

RedisのデータモデルはKey・Value形式で、複雑なデータ構造[5]を作るのは大変です。Study Typerでは以下のように設定しています。

```
# 英単語の集合

words:1, words:2, words:3  # words:[rank] キーをSorted Sets型のデータで定義

words:1 = [ attack, cook, ask, come, ... ]  # 例: ランク1の英単語のリスト

# 単語の個別情報

word:attack, word:cook, word:boondoggle  # word:[word] キーをHash型のデータで定義

word:attack = { rank: 1, ja: 攻撃する }  # 例: attackという英単語の情報
```

Redisは検索向きではなく、キーを指定して特定のデータを取得することしかできません。必要な値がピンポイントで取れるように、あらかじめデータとして持たせなければならないのです。

このような工夫は必要ですが、Redisを使ってみて、結果的に良かったんじゃないかと思ってます。実際に英単語を取ってくるのも早いです。

## 15.6　英単語データ

英単語データはejdic-handというフリーの辞書を使っています。英単語音声データはproject shtookaから無料データを探して使いました。英単語の難易度を設定するためのデータは某辞書サイトをスクレイピングしています。

これらの別々の情報を結びつけてRedisに保存するバッチスクリプトはPythonで書いています。コーディングは簡単でしたが、フリーのデータを探すのが大変でした。

S3とCloudFront

静的コンテンツの配信はAmazonのS3とCloudFront（CDNサービス）を使いました。

最初はS3だけで良いかなと思っていましたが、公開した後で英単語の音声再生に間がある（音声ファイルの読み込みが遅い）という問題が起きました。この問題を解消するために、CloudFrontも使うようにしています。その結果ちゃんと一瞬で英語音声が鳴るようになりました。

CloudFrontの設定はそんなに難しくなくて良かったです。転送するデータ量が多いときはCloudFrontを使ったほうが結果的に料金が安くなるらしいです。

## 15.7　Reactの実装

ここからが本番です。僕が大好きなフロントエンドの実装のお話です。楽しみですね。

### Flow

今回は「Flow」[6]を使いました。TypeScriptのように漸進的型付けを可能にします。最初は戸惑いましたが、Reduxリポジトリにあるtodos-flowというTODOアプリのFlow版サンプルを参考にしたら良い感じになりました。

初めて漸進的型付けで開発しましたが、かなり良さそうです。慣れるまではどう型を付けたら良いか分かりませんが、やるうちにコツが掴めます。コードが肥大化して自分が書いたコードを忘れるくらいになってから見ると改めて型の偉大さがわかりました。エディターの補完もバリバリ効くのがGoodです。

最近は流行としてFlowをTypeScriptに移行する流れが強いらしく、僕も移行を検討しています。

### Redux

Reduxという状態管理ライブラリーを使っています。React界のデファクトです。ユーザーが打ち込んだ文字や英単語のデータを管理しています。正直この規模のアプリでは過剰ですが、これからスケールアップする可能性と、Reduxを使い慣れているので使っちゃえという感じでした。Reduxは慣れてきたのでちゃんとした設計になっていると思います。非同期処理はthunkでやっています。

### CSS

まだ慣れていないstyled-componentsを使わず、less, bootstrapで手軽に実装しました。UIに時間かけたくなかったので。次のプロダクトはCSS in JSに挑戦したいです。

### Prettier

Lint系はPrettierを使いつつairbnbのコンフィグをextendsして一部修正しています。Prettierは最高です。1行に入る文字数を考えて整形してくれます。コードフォーマットについて考えたり議論する時間は減りそうです。迷える子羊たちよ、Prettierに従え。

### Jest

Jestというテストツールを今回初めて使いました。デフォルトでカバレッジを取得できるため使いやすいです。今回は純粋関数でテストがしやすいReducerのみテストしました。UIは頻繁に変更する予定なので、

Enzymeを使ったUIテストはやりませんでした。

## 15.8　最後に

　このように今回のアプリ開発では「最近のフロントエンド開発」を意識して、ナウい構成をキャッチアップしたつもりです（CSS周り以外）。新しい技術に挑戦しながらの開発で、ダラダラと1年くらいかけました。ですがそれを苦痛とは思わないほどフロントエンドの新しい技術は面白いものばかりでした。

　また、タイピングゲームという題材はシンプルで良かったと思っています。題材選びが良いと続けやすいですね。

　運用環境も、初めて使うCloudFrontなど挑戦ばかりでした。Webアプリの個人開発でデプロイまで辿り着けたのはこれが初めてです。

　以前はフロントエンドのフの字も知りませんでしたが、ここまでできるようになりました。独学中心ですが、研究室の先輩からの教えや、アルバイトの経験が役に立ちました。

　個人的な感想として、JavaScriptのライブラリーの入れ替わりスピードは尋常ではない気がします。次々と新しいものが登場しては、次々と廃れていきます。そんな中で僕たち人間に必要なのは、表面的な使い方だけを覚えるのではなく、その技術が何をしているかをしっかり理解することだと思いました。そのことを意識して公式ドキュメントのGetting StartedやCore Conceptに目を通しています。

　自分の性格的に、ただ動けば良いというよりも「コードがどうなっているのか」「設計がしっかりしているか」が気になり、突き詰めようとしてしまいます。そのためインプットばかりを行っているので、もっとアウトプット速度を早めたいなと思いますね。いや、フロントエンド技術をインプットするのは、本当に面白いんですよ。

　何か感想や意見がありましたら Twitter「@macinjoke」やGitHubのIssueにドシドシお願いします。こういうシステム構成の方が良いよーなどのツッコミも欲しいです。[7]

```
まっきー / @macinjoke

都内大学院生。研究室に週3で泊まるなどの奇行を繰り返す日々。Webを中心に趣味や
アルバイトで開発をしている。Reactに出会ってからフロントエンドの深みにハマっていく。
プログラミングと筋肉は裏切らない。

http://study-typer.macinjoke.com
```

1. http://study-typer.macinjoke.com
2. タイピングアプリなので、スマートフォンには対応していません。
3. http://dokku.viewdocs.io/dokku/

4. https://github.com/dokku/dokku-redis
5. RedisでRDBのように複雑なデータを扱う例として https://redis.io/topics/twitter-clone という公式チュートリアルがわかりやすかったです。
6. https://flow.org/
7. 本稿は執筆者（@macinjoke）によるブログ記事『英語学習用タイピングWebアプリをReactで作った話』を加筆・修正して寄稿したものとなります。

# 第16章　「このままでいいの？」駆動のサービス開発

　地域の魅力をクイズで見つける・発信できるWebサービス「Korette」[1]運営者のHirozです。プログラミング経験がゼロの状態から、独学で「Korette」を創り上げました。開発のきっかけ、アイデアの出し方、リリースまでの葛藤、得た学びをご紹介します。「このままでいいの？」という気持ちを胸に秘めている方々に向けて、私の経験をお話します。

## 16.1　地域の魅力をクイズにする「Korette」

　Korettoでは、観光地にまつわる歴史や音楽、人物などをクイズにすることで、観光地の魅力を楽しみながら知ることができます。

図16.1: Korette


　題材となる観光スポットのデータには、自治体のオープンデータ（約1万件）を活用。2017年12月にリリースした後、2018年3月には東京都主催の「東京都オープンデータアプリコンテスト」で準優勝（優秀賞）。2019年2月には、内閣府主催の「第三回RESASアプリコンテスト」で三菱総合研究所賞を頂きました。

しかし、私の本職はエンジニアではありません。プログラミング経験はゼロでした。その状態から独学で約9カ月かけてKoretteを開発しました。サービスを作ろうと思ってからリリースするまでの日々。想像以上に大変なもので、何度も挫折し、諦めかけました。

## 16.2　諦めかける日々との戦い

> 「なぜやろうと思ったのですか？」
> 「なぜ続けられるのですか？」

Webサービスをリリースしてから、よく受ける質問です。

私が今日まで運用を続けることができたのには理由があります。

・「自分の性に合っていた」という理由：パソコンを弄ったりするのが好きだった。プログラミングをやってみたら楽しかった。

・開発手法に関する理由：毎日コツコツ作業するようにしたことで習慣化できた。自分が持っている知識・スキルをなるべく流用して学習コストを抑えた。

しかし、それだけではありません。もっと大きい、根源のような理由があります。

エネルギーの源泉

それは、自分の中の「**このままでいいの？**」という気持ちです。みなさんは、ふと「**このままでいいの？**」と思ったことはありませんか？自分に、他人に、組織に、そして世の中に。

自分の中に「**このままでいいの？**」を持つということはどういうことなのか。なぜ私は「**このままでいいの？**」を持つようになったのか。どうやって「**このままでいいの？**」を解決するアイデアを思いつくのか。

私の話が、これからWebサービスを開発・運用する方の後押しとなれば嬉しいです。

## 16.3 「このままでいいの？」を持つということ

「中年の危機」という言葉を聞いたことがありますか？人生も半ばに差し掛かった頃、自分の生きる意味を問い直さずにいられない、心理的葛藤を表した言葉だそうです。カナダの精神分析学者エリオット・ジャックは、これを「ミッドライフ・クライシス」と名付けました。なんと8割以上の人がこのミッドライフ・クライシスを経験するとも言われ、欧米では既に社会問題として認知されてます。私はどうやら、この中年の危機に遭遇したようです。

自分自身に対しての「このままでいいの？」

10年ほど前、仕事で大きなプロジェクトに関わることになりました。がむしゃらに仕事をして、なんとか食らいつく日々。プロジェクトがひと段落しようとする、ちょうど自分が35歳になる頃。突如として、自分の中に疑問が湧き起こりました。

> 「このままでいいの？」

上り坂を必死で登ったら、急に平らなところに出て、道に迷うような感覚。自分はいったいどこへ向かっているのか。急に暗いトンネルに入って、出口が見えなくなる感覚。進めば進むほど、トンネルの先が狭まり、自分の可能性が閉ざされていくような恐怖。

これまで自分の中にあった成長の実感がなくなってしまった感覚。成長の踊り場。これは渇望？それとも焦り？その恐怖から、自分への疑心暗鬼が芽生えました。

葛藤の渦

会社の中で成果を出した気でいるが、井の中の蛙なんじゃないか？

自分の強みは何だ？　強みを生かして社会に対して価値を生むことができてるのか？

会社の肩書がなかったら、価値を生み出せない人間、無価値な人間ではないか？

年を取れば取るほど、世の中に価値を生まない人間になっているんじゃないか？

会社にしがみついて、働かないオジサンとして過ごすつもりなのか？

子供に対して「ああしろ」「こうしろ」と偉そうに言うけれど、自分はそんなに立派な人間なのか？

自分にできていないこと、自分が諦めたものを押し付けてるだけじゃないか？

父親として、背中を見せられるような生き方をしているのか？

今のお前の丸まった背中を見た子供は、一体どう思うんだ？　その背中を見せ続けるのか？

これまで自分と向き合ってこなかったツケが回ってきたかのように、葛藤の渦へと飲み込まれていきました。

## 答えを探す中で出会った個人開発

答えを探すため、最初はビジネス書を読み漁りました。ビジネスモデルの作り方、アイデアの出し方、マーケティング、経営戦略、チーム運営、組織論、マネジメント、さらには禅の教えまで……。目に留まったものはすぐに購入し、年間で200冊もの本を読みました。

本を読むうちにインプットだけでなくアウトプットも必要だと考えました。ネットの告知を見て、個人が主催する勉強会に参加しました。しかし、残念なことに、札幌では満足できる勉強会が見つかりませんでした。

私は「課題に対してビジネスアイデアを出し、ビジネスモデルとして持続可能な形にするまで」を知りたかったのです。そこで私は、自ら定期的に勉強会を主催しました。学びたいことが学べる上に、多くの人との出会いや交流もありました。

ただ、自分の中に沸いた「**このままでいいの？**」という疑問は解消しませんでした。むしろ、年齢を重ねるごとにその闇は深くなっていったのです。

机上の学習ではなく、実感を得られることがしたい。自分の好きなことや得意なことを活かして、社会に何か価値を生み出す活動がしたい。そんな悩みを抱える日々を過ごすうちに、Korettteとの出会いが訪れます。

## 自分の住む街に対しての「このままでいいの？」

Korettteは「地域の魅力をクイズで見つける・発信できる」Webサービスです。ジャンルとしては観光系が近いと思います。

私は、観光地である札幌の在住ですが、観光業とは関わりのない仕事をしています。当初はこの領域に踏み込む想定はなく「転職のこんなはずじゃなかった」を解決したいと考えていました。そんな私がKorettteの開発を決めたのには、きっかけがありました。

## 観光に行くからおすすめ教えて！

遠方に住む知人が札幌へ旅行で来る前に言ったひと言でした。

> 「札幌行くんだけど、どこ行ったらいい？おすすめの場所を教えて！」
> （えっと……時計台は「日本三大がっかり」だし、テレビ塔も別に感動するものでもないよなぁ。羊ケ丘展望台は、原っぱとクラークの像しかないし……）

知人を満足させられるような、気の利いた返事ができませんでした。みなさんは、もし自分の住んでいる町のおすすめの場所を聞かれたら、答えることができますか？

## 身近にあるのに知らない魅力

「私の街なんて、なんにもないよ～」と答えてはいませんか？自分の住んでいる街の魅力に関して、実は知らない場合も多いのではないでしょうか。

どんな街にも素晴らしい魅力はあるものだと、Korettеを作る過程で私は身をもって体験しました。しかし、住んでいる当人は、身近にある地域の魅力を「当たり前」と捉えてしまいがちです。

このような状況では、街が持っている隠れた魅力は、発掘・発信されません。観光に行きたいと思っている人が、魅力的な情報にたどりつけない状況に陥ってしまいます。

観光しようとしてネット検索する際に、こんな経験はありませんか？

・「情報が多すぎて、欲しい情報を探すのに手間がかかった」

・「必要な情報を見つけられなかった」

・「色々な情報があるようで、実はどれも同じような情報ばかりだった」

地域の人が身近にある魅力を知らないこと。観光するときに、画一的な情報が溢れて、欲しい情報を得られないこと。このふたつは繋がっている。そう思いました。

さらに、自分の住む街の魅力を知らない状況は、長期的にも不幸なことだとも考えました。住んでいる人が大切だと思わなければ、街の魅力は保存されずに失われていきます。地域住民が歴史ある建造物を「価値がある」「大切にしたい」と思わなかった場合どうなるでしょうか？

「古い建物なんかさっさと壊して新しいビルを建てよう！」

このように、歴史ある建造物が無策に取り壊され、街の魅力は失われてしまうかもしれません。魅力を失った街には、人もお金も集まらなくなります。地域に住む人たちは、自分の街への愛着も仕事も無くなり、街を出てしまうでしょう。

街が静かに死んでしまう。私の中に、自分の住む街に対する「**このままでいいの？**」という気持ちが沸き上がりました。

## 地域に住む人と、共に観光を創る

自分の住む街や故郷の街が、子供たちの時代まで、ずっと魅力的な街であり続けて欲しい。魅力を知ると、好きになる。好きになると、その魅力を誰かに伝えたくなる。魅力が伝わると、交流や来訪が行われる。やがて地域資源への愛着が生まれて、次世代まで残したくなる。この好循環を生み出すことで、地域資源の発掘・保存、さらには地域経済の活性化へと繋げていくことが大切です。

「観光＝観光客と観光業」だけで捉えるのではなく、地域住民を含めたものにする。観光で得られる利益を観光業以外の産業にも広げていき、地域資源や地域の住む人たちも持続的に潤うようにする。そうい

う未来を描き、作っていくべきではないでしょうか？

「おすすめの場所を教えて」をきっかけに、住んでいる街や故郷に対して**「このままでいいの？」**という気持ちが沸き上がり、貢献したいという気持ちにだんだんと変化していきました。

自分の内に沸いた2つの「このままでいいの？」に答えを出したい

・自分に対しての「このままでいいの？」

・住む街に対しての「このままでいいの？」

この2つに、どう答えを出すか。

自分が好きなのは、ITを使うこと、新しいアイデアや戦略を考えること。そして、決まっていない物事を進めること。ストレングスファインダーの診断結果には、着想・戦略性・最上思考・分析思考・未来志向の資質があると書かれていました。

そうだ、自分が思う世界を実現するための土台となるサービスを作ろう。自分でビジョンを定義し、ストーリーを立案、実行していこう。ITを活用したサービスを創造しよう。ITを使えば、個人でもたくさんの人の課題を解決できる。小さなチームでも大きな仕事ができる。世界を変えた人もいる。費用をかけずに小さく始めることができるし、自分が知識やスキルを習得すればするほど、結果として現れやすい。

そう考えた時、これまでの葛藤の中でバラバラに得てきた知識やスキルがひとつに繋がるような、このために今まで学んできたんだと思うような繋がりを感じました。

その後は、約9ヶ月間のサービス開発する日々を送りました。Qiitaに「中年の危機ど真ん中のオッサンがWebサービス作ってみた」という記事を書いたので、是非お読み頂ければと思います。

「このままでいいの？」の答え

私はちょうど40歳になります。時間で言うと、人生の正午にあたるのだそうです。身体の衰えや見た目に老化を感じるようになりました。同年代の訃報を聞く機会も増えました。

私の中に沸き上がった**「このままでいいの？」**に対する答えは、まだ出ていません。例え失敗しても、人生で1度くらい、自分でやるべきだと思ったことの旗を立て、仲間を募り、事を成そうとしても良いのではないか。今はそんな気持ちでいます。

これからもっと色々な事を経験し、人と出会っていくうちに、変わっていくかもしれません。いえ、変わらないかも知れません。ひょっとしたら単なる中年のオッサンの自己満足の葛藤として、何事も無かったように人々から忘れられていくかもしれません。

それは時が証明してくれるでしょうし、それしか証明する方法はないと思います。いずれまたどこかで、その後の進展をお伝えしたいと思います。

## 16.4　アイデアはどうやったら思いつくのか

「**このままでいいの？**」の次に直面する壁は、「それをどうやって、どういう手段で解決するか」です。Koretteの場合、住む人にとって当たり前すぎてありがたさが分からない地域の魅力を、どうやって認知して貰うか、どうやって世界に発信するか。

「当たり前のものを大切に」と、単に呼びかけても伝わりません。情報が溢れるこの時代に、良いものを「良いものです」とそのまま伝えても訴求しません。何とかして地域の魅力に気づいてもらう。利用者にとって入手しやすい形で届けて、行動してもらう。

そこで、私は解決の手段として「IT×クイズ」というアプローチを考えました。クイズという形で地域の魅力を切り出し、訴求力の高い形にして届けます。また、ITを使うことで、低いコストでたくさんの人に伝えることが可能となります。

よく「アイデアマンですね！」「クイズ好きなんですか？」と言われますが、Koretteでクイズを用いたのは、課題を解決する手段としてクイズが有効だと思ったからにすぎませんでした。

「アイデアとは既存の要素の新しい組み合わせ以外の何者でもない」

ジェームズ・W・ヤングというアメリカのコピーライターが、70年も前に著書の中で書いた言葉です。アイデアはセンスよりも、無理やり作り出すものなのです。

「何とかしたいこと × ○○○」を書く

アイデアを出すノウハウについて、さまざまな本が出ています。「アイデアが思い浮かばない」「自分にはセンスが無い」と思っている人でも、サービスのアイデアは作り出せます。

私の場合は「観光（地域の魅力発信）×IT×○○○」の形で、○○○に入るものをひたすら書き出しました。付箋に書き込み、そこから取捨選択するというアナログな手法です。

100個出して、10個まで絞り込みます。絞り込む時に気を付けたのは「直接的に人の役に立ちそうなものにしないこと」です。

目先で誰もが必要だと思う事や、誰かの役に立つものは、きっと競合が現れるに違いない。長い目で見て人の役に立つようなものや、企業も直近では儲からず参入しそうにないものである点がポイントです。

残った10個については、競合や類似事例をネットで調べて企画を考え、しっくりくるものを選びました。こうして「観光（地域の魅力発信）×IT×クイズ」というアイデアが出来上がりました。

単なるクイズサービスだったら、レッドオーシャンになるでしょう。デザイン力も開発力も低い私には、勝ち目はありません。しかし「観光×クイズ」なら、少しだけニッチなポジションを確保し、市場の隅っこで大きくなって

いけるのではないかと考えたのです。

クイズで観光地の魅力を発掘し・発信する

クイズと聞くと、お遊びのイメージが強いかもしれません。しかし、実はすごい力を備えています。それは「魅力をリフレーミングするチカラ」と「人を惹き付けるチカラ」です。

クイズであれば地域の魅力を様々な形に抽出して、たくさんの人に届けることができます。例として、Koretteに実際に投稿されているクイズを出してみましょう。

> 「京急立会川駅の前にある坂本龍馬像が履いているのは何でしょう？」

東京都・品川区のクイズです。坂本龍馬と言えば先進的なイメージですから、西洋の履物「ブーツ」を想像するかもしれません。しかし、正解は「草履」なのです。黒船が来航した幕末の龍馬がモデルになっているため、ブーツではなく草履なのだそうです。

クイズにすることで、単に龍馬の銅像があるというだけでなく、通常は目が行きにくい足元に注目することができ、龍馬にまつわる歴史も知ることができます。ただ「品川にある坂本龍馬像は草履を履いている」と伝えられるより、記憶に残る気がしませんか？

このようにクイズにすると、対象を様々な視点でリフレーミングし、その魅力を抽出することができます。また、人は質問されると内容に興味を持ち、つい答えたくなります。答えを知って納得すると記憶に残り、楽しくて誰かに同じ質問をしたくなるのです。

クイズの持つ力は、回答者だけでなく、クイズを作る人にも及びます。クイズのネタを探して地域を見渡すと、見慣れた景色から新鮮な魅力が浮かび上がります。対象の魅力を抽出しやすくなり、自分自身も詳しくなることができます。是非1問だけで良いのでクイズを作ってみてください。きっと、その楽しさにハマるはずです。

そんな人がたくさん増えたら、街の魅力を大切にしよう、次世代に受け継ごうという人も増えます。街の魅力をデータベース化すれば、観光ボランティアや地域の子供への教育といった用途の利用も可能でしょう。

私は、子供を持つ父親でもあります。子供のために、自分たちの住む街の維持発展に、少しでも貢献できるサービスを目指しています。

## 16.5　サービスを開発したいと思っている人へ

### やることの9割は思うようにならない

リリース前は「リリースするまでの辛抱だ」「リリースさえすればユーザーがどーんと来るはず」と思っていました。しかし、実際にはリリース後も「諦めず、地道に、目の前の使ってくれる人に感謝しながら、少しずつ利用者を増やしていく」という泥臭い日々が待っていました。

やってみて分かったことは、「やることの9割は思うようにならない」ということです。予想外に人気が出たりバズることもあれば、練りに練った施策が空振りに終わることもあります。自分ではどうすることもできない、追い風や向かい風のようなものを感じます。日頃の行いを正し、徳を積む活動をしたくなることもあります（笑）。

これまで辿ってきた道、今自分が立っている場所は、当初思い描いた場所とは異なります。これから向かう道も、きっと辿りついてみたら、違う場所に立っているでしょう。

仕事だったら、嫌になってメンタルを病むかもしれません。しかし、私の場合は、自分のやりたいこと、やるべきことですから、辛いと思う気持ちはありません。私がサービスの開発や運営を通じて「**このままでいいの？**」を解決する旅を続ける限り、開発や運営に飽きることはないでしょう。

### 色々な中毒要素があなたを迎えてくれる

やることの9割は思い通りにならないですが、日々の中に喜びや満足を感じることはたくさんあります。

私の活動やブログを見た方から「何かを始めるのに年齢なんて本当に関係ないと思えた」「サービスへの思いや可能性、行動力にすごく良い刺激を受けた」と言って頂ける機会がありました。私のブログに刺激を受けて「自分もサービスをリリースしました」と伝えてくださる方や、私のサービスを応援し「手伝いましょうか？」と声をかけてくださる方も現れました。

サービスのUIを提案してくださる方。PR用のチラシを作ってくださる方。そして、率先してクイズを投稿し、機能の不具合や改善点を教えてくださる方。さらに、スマホアプリを開発してくださる方。嬉しくて涙が出そうになります。

このように、たくさんの人に恵まれ支えていただいているのは、自分のなかに「**このままでいいの？**」を持っていること。そして、それをサービスのビジョンとして掲げている面も大きいのかもしれません。

サービスを開発することで、様々な知識や経験を得ることができます。プログラムの技術、サービスを動かすためのインフラ、サービスを維持するノウハウ。プロモーションや関係者との連携といった、ビジネスを行う上で必要な知識や経験。

ほかの個人開発者や自分と同じビジョンを持つ人など、普段の生活では出会えない人との繋がりを得ることもできます。本や勉強会では得られない、自分でサービスを作って運用しているからこそ得られる素晴らしい資産です。

## たくさんのサービスを作るのか、ひとつをじっくり育てるのか

個人開発者には、ひとつのサービスをずっと続ける人と、たくさんのサービスを作る人の2つのタイプがいると思います。どちらが正しいということはありません。

私は、自分の中に明確なビジョンや目的がなければ、行動できない、続けられない人間なので、ひとつのサービスをずっと続けています。

一方で、たくさんのサービスを開発する人も、その人なりに共通のテーマを持っているように感じられます。自分なりのテーマや価値観を持つことはモチベーションに繋がりますが、最初からテーマを明確にできるかというとそうでもないと思います。「自分のテーマや価値観は何だろう」と意識しながら、まずは行動すると良いでしょう。

## 16.6　Koretteのこれから

Koretteのリリース後、ありがたいことに地域振興活動に積極的な企業様や団体様とのコラボレーション企画が始まりました。2018年5月には、北海道弁PRキャラクター「やべーべや」とのコラボレーションが実現。「方言」をテーマとしたクイズで、多くの方から「北海道により関心や愛着を持つ機会になった」とご好評をいただいています。

また、リリース当初はWebサイト形式のみでしたが、Koretteの趣旨に賛同いただける方との協働プロジェクトという形式で、1周年を迎える2018年12月にスマートフォンアプリ（iOS/Android）の配信を開始しました。

さらに、活動を推進するために、2018年12月にKoretteを運営する団体として「CulutiVision」という組織を立ち上げました。Koretteが目指す地域の持続的な発展の支援のためには、自治体や観光協会、企業などとの協力関係や支援が不可欠です。組織の設立を、個人での運用からの脱却を図る足がかりにしたいと考えています。

CultiVisonは、Vison（将来）をCulti（耕す）という意味を込めて造った言葉です。農業は時間をかけて土を耕し、中長期的に実らせる。同じように、私が自分の住む街に対して感じた「**このままでいいの？**」を解決し、地域が持続的に発展していくお手伝いをしたいという想いを込めました。団体の設立により、ビジョンや進む方向を明確にして発信していくつもりです。

楽しみながら自分の住む街を知り、好きになる。「好き」が溢れて繋がって、たくさんの人との繋がりや交流が生まれ、地域が持続的に発展していく。そのためのプラットフォームとして、Koretteを成長させていけたらと考えています。

## 16.7　あなたの「このままでいいの？」は何ですか？

私がWebサービスを開発し、今日まで続けている理由は、自分の中に沸いた自分に対する「**このままでいいの？**」、そして社会に対する「**このままでいいの？**」に対する答えを出すための活動だからです。

「**このままでいいの？**」駆動開発です。

いつか中年の危機を迎える10代・20代の皆さんへ。年齢を重ねると、私のように「**このままでいいの？**」と思う日が来ると思います。世の中に役立つ何かを生み出す、Webサービスで実現する、というのは、限られた資金・時間の中で有効な手段です。まずは、一歩を踏み出してみましょう。それだけで、たくさんの人よりも前に進んでいるはずです。

中年の危機を迎えた30代、抜け出せずにいる40代以上の皆さんへ。何かを始めるのに、遅いということは無いと思います。みなさんは、私よりももっと上手く、何かを成し遂げる事ができるはずです。Webサービスの提供は、費用も抑えることができ、後でやり直すことも可能です。まずは、一歩を踏み出してみましょう。それだけで、たくさんの人よりも前に進んでいるはずです。

最後までお読みいただき、ありがとうございました。私のお話が、皆さんが少しでも早く前に進める手助けとなれば幸いです。

```
Hiroz / @hrz31


札幌在住。クイズで観光地の魅力をみつけるWebサービス「Korette」の開発者。
エンジニアではない、普通のサラリーマンだったが、ゼロからプログラミングを学ぶ。
2018年12月にはサービスを運営する団体「CultiVison」を設立。


https://korette.fun/
```

1. https://korette.jp/

# 第17章　使われなくてさみしい問題を解決する（しない）方法論

　ぼっちスタートアップでニッチなWebサービスを作っている、ほげにし（@kame_f_no7）です。代表作は、青空文庫の短編が毎日メールで届く「ブンゴウメール」や、トーナメント作成サービス「THE TOURNAMENT」など。これまで一番悩んだのは、作ったものが誰からも使われなくてさみしいという問題でした。この問題を解決するために編み出した方法論を紹介します。

## 17.1　Webサービスの死因

これまでの経験をもとに、わたしが考える「サービスの死」とは以下の3つの状態です。

【死因①】ノーコンテンツ：コンテンツがメインの価値になるサイトで、コンテンツが圧倒的に少ない OR あっても全然更新されてないなどで、有用なコンテンツがない（あるように見えない）。

【死因②】動かないシステム：システムのエラーなどで、本来の機能を利用できない状態が放置されている。Twitterログインが機能していないなど。

【死因③】サイトの消滅：ドメインが切れてて、そもそもサイトにアクセスできない。サーバーを解約して、もう何も動いていない。

③が完全な死ですが、①と②みたいなサイトを見たときも「実質的には死んでいる」と言えるのではないでしょうか。個人開発でWebサービスを作ったものの、全然ユーザーが集まらなくて自然消滅、というのはよく聞く話です。

## 17.2　個人開発サービスの一生

わたしが個人開発で経験したWebサービスの一生はこんな感じでした。

1．サービスをリリースしたけど全然使われず、がんばって自分で投稿したりする
2．がんばって宣伝したりするけどやっぱり使われず、自分でも投稿しなくなり放置されだす（①ノーコンテンツ）
3．ライブラリーや連携外部サービスのアップデートに追従しなくなり、サービスが動かなくなる（②動かないシステム）
4．ドメイン代がもったいなくて、更新をやめる（③サイトの消滅）
5．突然の死！！！

書いていて動悸が激しくなってきました。恥を承知でさらすと、わたしのサービスはこんなのばっかりです。

## 17.3　失敗例：book loves music（ブック・ラブズ・ミュージック）

図17.1: book loves music


　読書にぴったりの音楽をおすすめし合うサービスです。[1]2011年くらいに作ったデビュー作です。ユーザーが本に合う音楽をセレクトすると、Youtubeのプレイリストとして再生できるというサービスでした。

　サイトコンセプトが受けたのか、リリース直後には複数のメディアに掲載されるなどプチバズを経験。しかし99％のユーザーは投稿されてるコンテンツをざっと見るだけでした。誰もコンテンツを投稿したりはしてくれませんでした。

　コンテンツがスカスカなので一度来てくれた方もそのまま離脱して戻らず、ますますサイトは過疎化の一途。結局半年ほどは自分で投稿したりして運用を続けたものの、力尽きてそのまま死を迎えました。2020年1月のドメイン更新をせずにこの世から消える予定です。

## 17.4 「バズか死か」の二元論を避ける

問題なのは、モチベーションが続く間にサービスが軌道に乗らないと、ほぼ間違いなく死んでしまうことでした。誰も使わない状態を自分ひとりで支えてるので、モチベーションが尽きると更新も止まります。

そして、どんなにテンション高くはじめても、モチベーションはいつか絶対に落ちます。波があるのは当然なので、モチベーションが下がってもサービスが死なない設計・運用にしたい。

わたしはつい「バズか死か」みたいな二元論で考えてしまいがちでした。そうではなく、死ににくい設計で長期運用を可能にできないか。モチベーションの波と付き合いながらのんびりサービスを育てられないか。

この発想の転換が「ビルド＆ゾンビ」です。意識低くサービスを延命させましょう。大きな死因が3つあるので、これを順番につぶしていけば必然的に死亡率を下げることができるはずです。

めっちゃバズらせる・ヒットさせる方法論ではなく、あくまでバズらないWebサービスでもさびしく孤独死するのを避けたい……。という後ろ向きなハックです。

## 17.5 【1】CGMではなくツール型にする

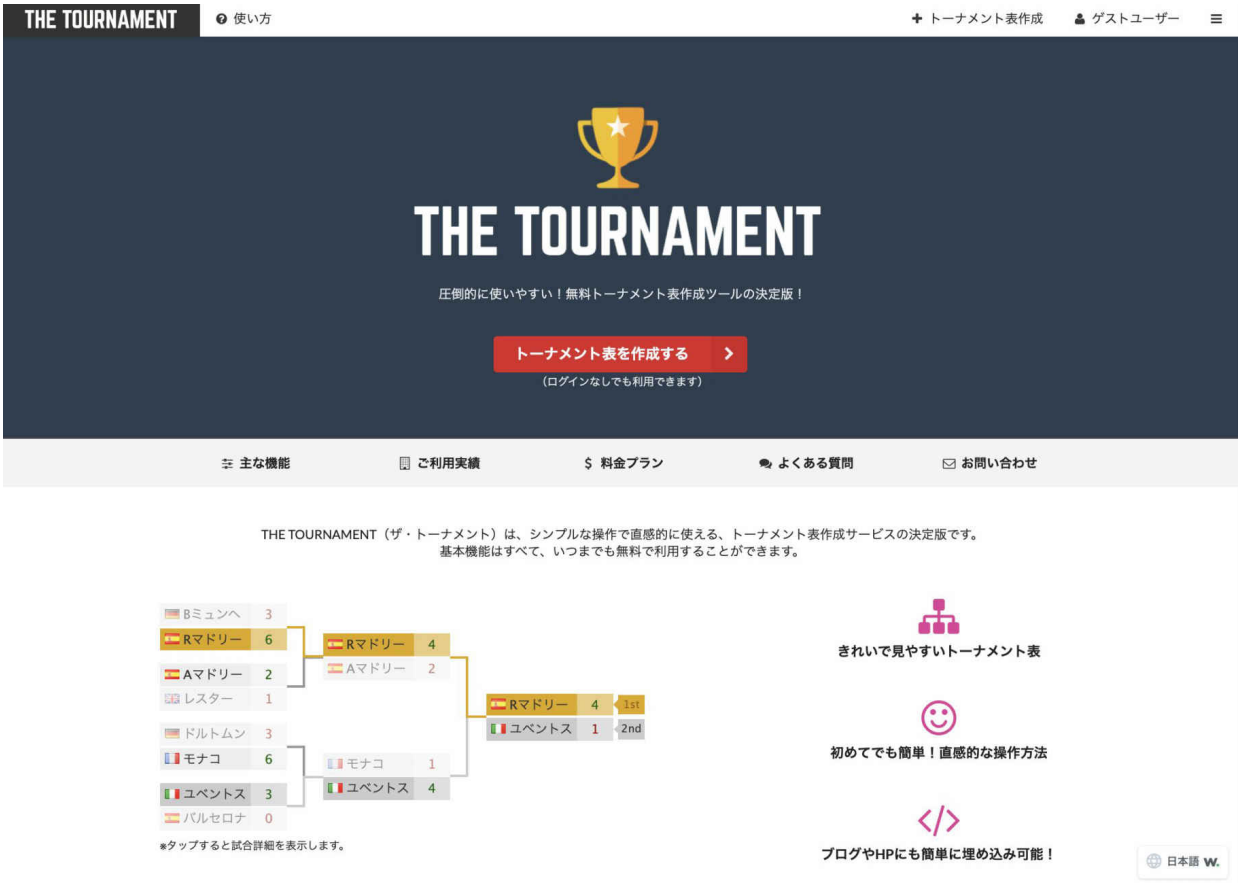
死因①の「ノーコンテンツ」はCGMだから起きる問題なので、CGMをやめるだけで解決します。ここで言う「CGM（Consumer Generated Media）」とは、「ユーザーの投稿コンテンツがサイトの中心的な価値となるようなサービス」のことを指します。有名どころではTwitterやCookpadを例に挙げることができます。

CGMはコンテンツが集まると圧倒的な価値が出るのですが、最初のコンテンツ集めが難しいです。コンテンツがないから人が集まらず、人が集まらないからコンテンツも増えない。いわゆる「ニワトリ・タマゴ」問題です。

そこで考えたのが「ツール型」のサービスにすることです。ここでツール型とは「他のユーザーが誰も使っていなくても価値を提供できるサービス」と定義してみます。

## 17.6　具体例：THE TOURNAMENT（ザ・トーナメント）

図17.2: THE TOURNAMENT




Web上で簡単にきれいなトーナメント表が作れるという地味なサービスです。[2]4年前にリリースしてから一度もバズったことはありません。しかし、検索で着々と利用者数を増やし、現在では有料の法人利用も増えてきています。

このサービスは典型的なツール型です。もう運営4年目なのですが、2年目まではごく少数の方に細々と使われる程度でした。誰も使っていなくてもサイトの有用性に影響はないため、時間をかけてゆっくり検索流入が増えるのを待つことができました。また、サイトが育つまではほぼ放置していただけなので、その間は他のサービスの開発に時間を使うことができました。

仮に数年くらい誰も使っていなかったとしても、死んだサイトには見えにくいです。CGMだと「寂れるとますます寂れる」という負のループに陥りますが、ツール型はその心配がありません。放置したままでも長期運用することができ、結果としてサービスをじわじわ成長させることができました。

## 17.7 【2】できるだけ何も作らない

死因②の「動かないシステム」ですが「極力何も作らない」という解決策に至りました。

放置したシステムは常に動かなくなる危険性を抱えています。Twitterログインの仕様が変わる、使っている言語やフレームワークのバージョンがサポートされなくなる、など。生きているサービスはこれらの波を乗り越えて寿命を延ばし続けるわけですが、どこかで開発者がこの対応を諦めると、そこでシステムとして死ぬことになります。

対応を諦めてしまうのは、開発者のモチベーションが下がった時です。サービスを作った直後、ユーザーにすごく使われている、売上が上がっている。そういった気合が入ってるときなら、問題が起きてもちゃんと対応できます。

問題は、リリースからしばらく経ったのに、サービスが伸びていないような時です。2年前に作って誰にも使われず放置してるサービスのRailsメジャーバージョンアップなんて、正直やりたくないです。何年もモチベーション保ち続けるのは現実的ではありません。モチベーションに期待するのは無理があるので、工数（＝めんどくさ）を減らす方向で考えます。

## 17.8　具体例：ブンゴウメール

図17.3: ブンゴウメール


青空文庫の名作小説を1ヶ月で読み切れるように小分けして、毎日少しずつメールで配信してくれるサービスです。[3]ありがたいことにリリース後にTwitterで大きな反響をいただき、メディアなどにも多数取り上げていただきました。

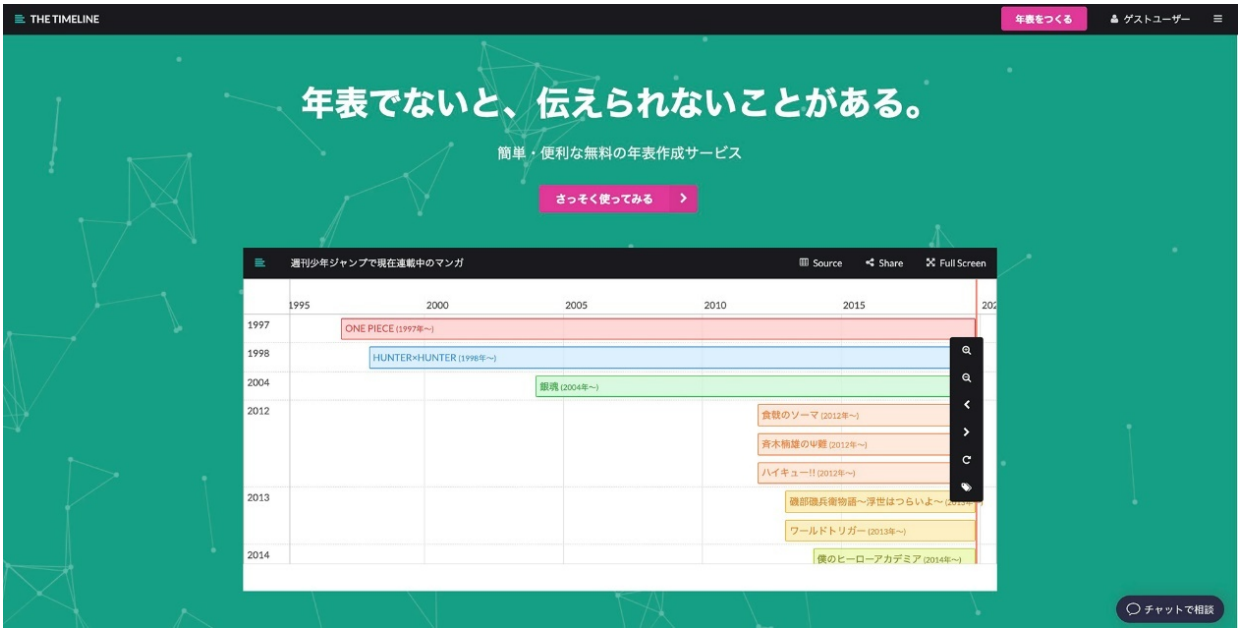
正直なところ、作っているときは、使われる自信なんて全然ありませんでした。ただ、自分がほしいサービスだったので、仮に流行らなくても長く使い続けられるように、極限までシンプルな作りにしました。

結果として、リリース時はフレームワークなど何も使わない、わずか150行のRubyスクリプトになりました。ローカルで実行すると、指定した作品を分割して1ヶ月分予約配信するというだけの超ミニマムプロダクトです。

ここまで小さいと滅多に不具合も出ないし、仮に何かあってもすぐに直せます。いわゆるMVP（Minimal Variable Product）という考え方です。MVPの状態で長期運用できるように設計しておくと、芽が出るまで放置して育てることができます。

## 17.9　具体例：THE TIMELINE

図17.4: THE TIMELINE




Web上で年表を作成できるサービスです。[4]開発から2年以上ほぼノーメンテですが、元気にご存命です。ありがたいことに最近になってじわじわ利用も増えてきました。

このサービスでは、年表のデータを自前で持たずにGoogleスプレッドシートに丸投げしています。スプレッドシートみたいな複雑なインターフェースを開発・維持する手間がごそっと省けています。

外部ツールならではの制約は多いので、将来的にはユーザーの要望を聞きながら改善するつもりです。その頃には本格的な成長フェーズに入って、モチベーションも上がっているはずなので、むざむざシステムを死に追いやったりはしないでしょう。

## 17.10 【3】絶対に赤字にはしない

死因③の「サイトの消滅」は、ドメイン切れやサーバーの契約解除など、お金の問題が関わってきます。わたしの経験談ですが、たとえ小さくても赤字の状態だと、どこかで支払うのをやめたくなってしまいます。というわけでこれに対する対応はシンプルで、絶対に赤字にしないことです。

なかでも、ドメイン切れによる死は何回もやってしまいました。ドメインを取得してサービスを作ったけど全然使われず、1年後の更新をせずにそのまま終了というパターンです。

サービスをサブドメインで始めると、この悲劇を簡単に避けることができます。自動更新してずっと使う主要ドメインを持って、新規サービスはそのサブドメインを使っています。

また、月数百円程度のサーバ代でも、解約したくなる日は来ます。数年前に作ってまったく使われてないサービスのためだけに払ってたりするからです。

わたしはHerokuやFirebaseを使うことが多いです。無料プランでも工夫すればけっこうな規模まで対応できます。特に最近はFirebaseが尋常じゃなく便利なのでおすすめです！

## 17.11　具体例：スマホde百人一首

図17.5: スマホde百人一首


　みんなのスマホを百人一首の取り札にして遊べるサービスです。[5]詠み上げ用の親機1台＋取り札用が最低2台と、3台以上のスマホを使うサービスです。100台の端末を並べると全ての取り札で遊べます。お正月にでもぜひどうぞ。

　これは完全に一発ネタのサイトです。作ってから4年以上放置しており、今後も改修の予定はありません。独自ドメインであればとっくにクローズしていました。

　サブドメインにしているおかげで、少なくともドメイン切れの心配はなく、いまだに動いています。毎年ドメインを更新するかどうかで悩むことがないのはとても快適ですね。

　そのため、長生きさせたいならサブドメインから始めてもいいんじゃないかなと思います。順調に育ってきたら、あとから独自ドメインに切り替えることもできます。

## 17.12　具体例：ゾラサーチ

図17.6: ゾラサーチ


青空文庫の作品を、目安の読了時間で検索できるサービスです。[6]前述のブンゴウメールの姉妹サービスとして作りました。

ブンゴウメールは超ミニマムプロダクトとして始めました。ありがたいことにユーザー数と機能要望も増えたため、Railsアプリとしてリニューアルしています。そのブンゴウメールに蓄積した青空文庫のデータを活用して、またもや自分がほしかったサービスを作ったのがゾラサーチです。

極力管理の手間は増やしたくなかったので、ブンゴウメールのRailsアプリ上で動かしています。Controllerを1つ追加することで実現しました。サーバも相乗りなので、追加の費用はかかっていません。

ゾラサーチは売上のないサービスです。しかし、こうした省エネ設計なので、サービスを閉じる必要は一切感じません。

## 17.13　死なないだけだと、ただのゾンビ

ここまで、3つの大きな死因を避ける方法を実践してきました。リリースしてすぐに「バズか」「死か」の二択になってしまう状況を避けることで、長期的にサービスを成長させるための土台を作ることができます。

しかし、これだけだと、死んではいないけど成功してるわけでもありません。死なないだけだと、ただのゾンビです。目前の死を先送りにできたら、改めてサービスを成長させる方法を考えてみましょう。

## 17.14　最後は確率論、いっぱい作ろう

サービスを延命させることで成長させるチャンスは増やせます。

「ブンゴウメール」は2年前にも一度リリースしていましたが、そのときは鳴かず飛ばずで、2年越しの再リリースでようやく反響をいただけました。コンセプトを見直したとか、文豪系ゲームが流行した影響だとか、色々な要因が重なっているはずです。最後は確率論なので、あきらめずにいっぱい作っていこうと思います。

スクラップ＆ビルドの繰り返しだと「バズか」「死か」で終わってしまいます。そうではなく「ビルド＆ゾンビ」で長寿サービスをいくつも作ることで、じっくりと芽が出るように育てられるのではないでしょうか。

好みが分かれるところではありますが、こんなやり方もあるよということで参考になればうれしいです。[7]

> ほげにし / @kame_f_no7
>
> ぼっちスタートアップでニッチなWebサービスを作っています。 名作小説をメールで小分け配信する「ブンゴウメール」、青空文庫を読了時間で検索できる「ゾラサーチ」、トーナメント作成サービス「THE TOURNAMENT」など。
>
> https://blog.notsobad.jp/

1. http://booklovesmusic.com/
2. https://the-tournament.jp/
3. https://bungomail.com/
4. https://the-timeline.jp/
5. https://karta.notsobad.jp/
6. https://search.bungomail.com/
7. 本稿は執筆者（@kame_f_no7）によるブログ記事『作ったWebサービスが誰からも使われなくてさみしい問題を解決する（しない）意識低い方法論』を加筆・修正して寄稿したものとなります。

# 第18章　サービス開発のハードルが高いあなたに！便利プログラム開発のススメ

　渋谷でWebエンジニアをしている仮想サーファー（@virtual_techX）です。仕事ではJavaで自社サービスを開発して、個人の趣味で便利プログラムを開発しています。自分や身の回りの人の作業を自動化するためのプログラムです。「個人で何か作りたいけど、何を作れば良いのか分からない」「Webサービスの開発はハードルが高い」という悩みを持っている方の参考になればと思います。

## 18.1　どんなプログラムを作ったの？

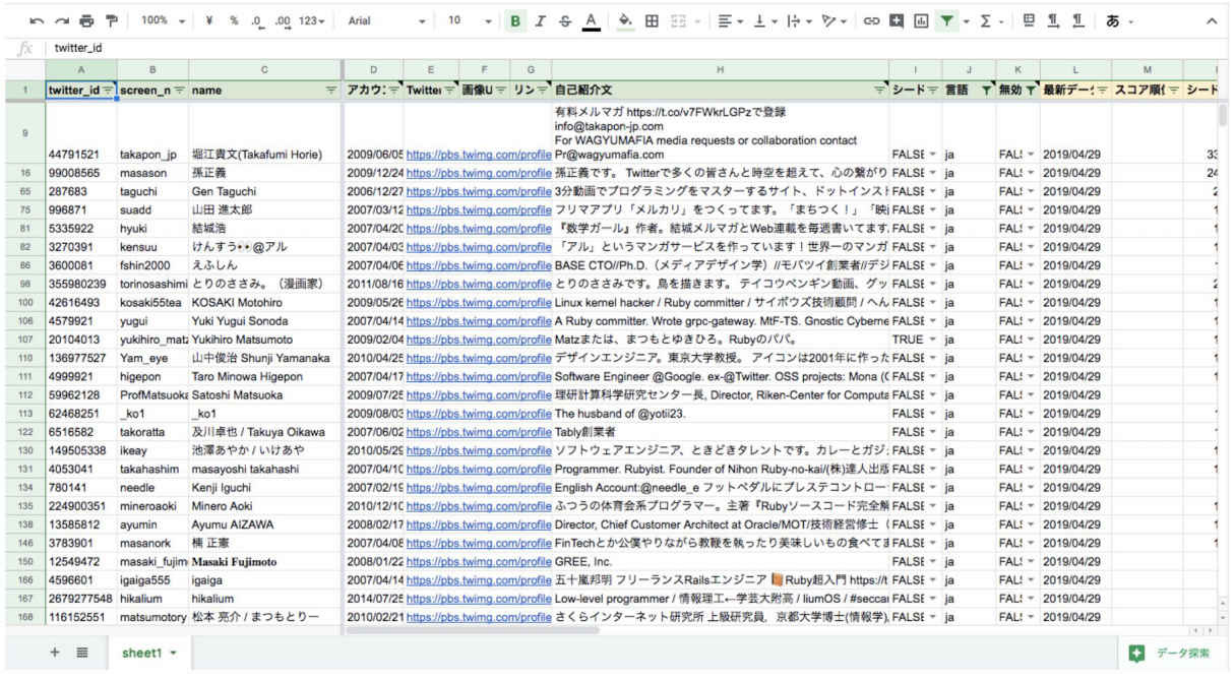
特定の界隈のTwitterアカウントのデータを取得してSpread Sheetに書き込み、分析レポートをはてなブログの記事として自動投稿します。このサービスは、ふたつの便利プログラムから成り立っています。

ひとつ目は、Webエンジニア・筋トレ・Youtuber・ブロガーなど、様々な界隈のTwitterアカウントの各種データを取得し、そのデータからどの程度の影響力があるアカウントなのかを算出し、界隈のインフルエンサーランキングを作成するプログラム。

ふたつ目は、インフルエンサーランキングのデータをもとに、ブログの記事を作成し、はてなブログに投稿するという機能。

これらふたつの機能を作成して、自動で定期的に動かしています。

図18.1: インフルエンサーの分析シート



## 18.2　なんで作ったの？

もともとは「個人でWebサービスを開発したいな！」という思いがあったのですが、アイデアが浮かんで来ないんですよね。顔が見えない誰かのためのWebサービスを開発するイメージが湧かなかったのも原因かもしれません。

「アイデアもないのにWebサービスを作ることはできないよなぁ……」「でも、業務外でも開発はしたいなぁ……」と考えた結果、「普段の作業を自動化するプログラムを作ってみよう！」と思いつきました。

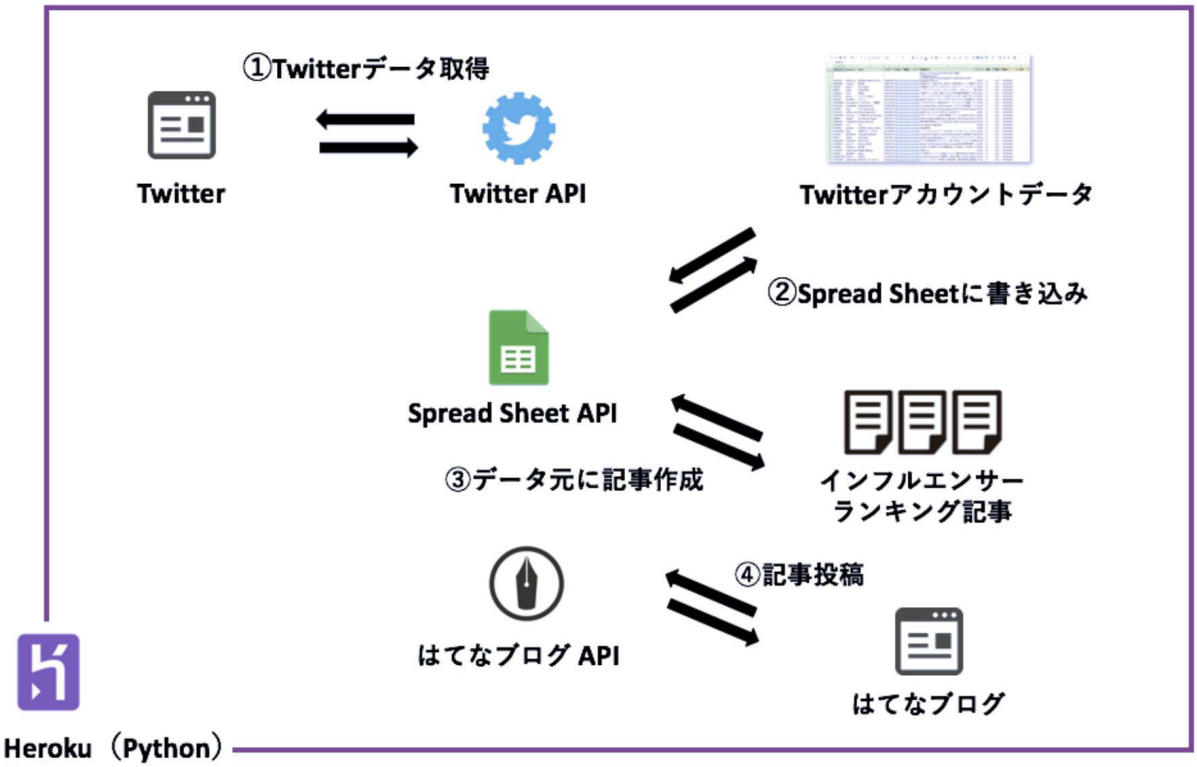
自分が時間を使っている作業は何だろうかと改めて考えました。するとTwitterでの情報収集とブログ記事作成に毎日1~2時間も費やしていることが判明。SNSマーケティングを学ぶために毎日コツコツとインプット・アウトプットしていたからです。

「これを30分に削減できれば、めちゃくちゃ時間の節約になる！」「1年で1万時間以上の削減効果だ！」と思い立ちました。

## 18.3　システム構成

以下のようなシステム構成で処理を実装しました。

図18.2: システム構成




利用したのは以下の言語・実行環境・APIです。

・Python

・Heroku

・外部API（Twitter API・Spread Sheet API・はてなブログAPI）

Pythonは未経験だったのですが、スクリプト言語で、コード記述量が比較的少なくて済むかなと考えて選択しました。

アプリの実行環境としては、簡単に設定できるHerokuを利用しました。Herokuで提供されているデータベース（MySQL）はあっという間に無料枠を超えてしまったため、代わりにSpreadSheetでデータを管理することにしました。

## 18.4　開発中のトラブルと反省

処理内容は「Twitter APIで取得してきたデータをSpread Sheet APIでシートに書き込む」と「はてなブログAPIで記事を投稿する」だけです。各APIの公式ドキュメントを読んで実装を進めました。

簡単に出来るだろうと気軽に開発していたら、リクエストを飛ばしすぎてしまい、APIのアクセス制限に引っかかってしまいました。過度のデータ収集や機械的な投稿は、各サイトの利用規約で禁止されています。節度のある振る舞いを心掛けましょう。

プログラムの開発自体は40時間程度の工数で完了し、約1ヶ月ほど自動で処理を動かしました。

## 18.5　便利機能を開発してみてよかったこと

開発してみてよかったことは3つありました！

①普段触らない技術に触れてスキルアップできた

普段業務で触っているJavaだけではなく、Pythonでの機能開発・デプロイができるようになりました。もし将来「Webサービスを開発したい！」と思ったときには、今回の学びが活きるはずです。業務外のちょっとした時間で経験できたのは良かったなと思っています。

エンジニアとして働いていく上で、スキルを伸ばし続けることは必須です。だけど会社の業務だけだと、どうしても限られたスキルしか伸ばせない。日常生活に活かせる便利なプログラムを実装しながら新しい技術を学ぶのはオススメです。

②社内外でスキルを活用できるようになった

今回紹介したプログラムではTwitterのAPIを利用しました。同じように、仕事で担当しているサービスについて、Twitterでの口コミを自動収集するプログラムを作りました。集めた口コミはSlackに自動投稿して、チームメンバーに届けます。

さらにその応用で、Slackでの定期連絡をプログラムに置き換えたりと、様々な作業を自動化し続けています。

最近は「いろんな作業を自動化してる人」として認知され始めてきたのか、オークションサイトの情報収集自動化とか、Slackでの通知全般自動化とか、社内外で作業を自動化してくれないかと依頼されるようになりました。

③noteとブログでプログラム記事を公開してマネタイズできた

最初は自分が便利に使うことをゴールにして開発しましたが、マーケティング職の友人に話をすると「自分もTwitterのデータをSpreadSheetに移して分析したいんだよね」と言われ、ブログとnoteで発信することにしました。

開発でやったことの概要をブログで紹介し、noteでプログラミングのチュートリアルを公開。公開から半年程度が経ちますが、その記事だけで月平均5,000円ほど収益が発生しています。何もしなくても毎月技術書を2冊買えます。最高。

## 18.6　振り返ってみて

> 「エンジニアは2度以上同じ作業をしたら自動化するべき。」

会社のエンジニア上司に言われた一言です。

その言葉にならって「自分の作業を定型化できるものはないかな？」「自動化する余地はないかな？」と、社内でも趣味の時間でも常に自動化の機会をうかがっています。

作業の自動化によって新しい技術が学べるし、周りの友人や同僚も自分の実装のおかげで楽ができる。そして、そのプログラムを欲している人に向けて発信することで、どこかで同じような作業を繰り返していた人も楽になる。プログラムを活用した人たちから、その楽になった分の対価として、ちょっとずつだけお金をいただく。こんな個人開発のあり方も、もっと広がってもいいかなと思います。

以上、ご精読ありがとうございました！記事内容で疑問やツッコミがある方は、ぜひ著者宛にご連絡ください。

仮想サーファー / @virtual_techX

渋谷のWebエンジニア。日常生活の作業を自動化するのが趣味。

エンジニアの働き方などをまとめた雑記ブログや、非エンジニアを対象としたプログラミングチュートリアルnoteも書いています。

https://www.virtual-surfer.com/

# 第19章　アクセスログから見えた時代の移り変わり

　WEB企業でプロダクト開発をしている @tky_bpp です。私が作ったサイトは「平成カウントダウン」[1]です。元号が変わるという歴史的なイベントを、個人開発を通して楽しんでみました。

## 19.1　サービス紹介

　2019年5月1日より施行された新元号「令和」までのカウントダウンサイトです。

図19.1: 平成カウントダウン


　カウントダウンを表示するだけの一発ネタです。改元の日程が公表された2017年1月にリリースしました。

　「令和」という名前は直前まで決まっていなかったので「平成の終わりまでのカウントダウン」という意味を込めてこのサイト名にしました。

　新元号になったいま、サービスとしての役目は終えています。

## 19.2　モチベーション

私は平成元年生まれです。改元の報道に衝撃を受けるのと同時に、人生を歩んできた「平成」が終わることに寂しさを感じてしまいました。

毎年の大晦日にはカウントダウンが恒例行事になっていることを思い出しました。改元も同じようにカウントダウンしてみたらよいのではないか。そんなことがきっかけでこのサービスを作ってみようと思いました。

また、サービスの内容としては極めてシンプルなので、勉強も兼ねてAWSを試すことにしました。仕事ではアプリケーションの開発やログを見るのがメインで、AWSに触れる機会が少ないため、どこかでチャンスを見つけて勉強したいと以前から思っていました。

## 19.3　AWSの構成

Route53（ドメイン）＋ AWS Certificate Manager（証明書） ＋ CloudFront（プロキシサーバー）＋ S3（ファイル）です。特に先進的なことはやっていません。基本的な設定が中心のため、AWSのドキュメントを参照しながら進めていきました。

設定が上手くいかないこともありましたが、トラブルシューティングを通して学びを深めることができました。他の開発者の方のブログなど、先人の知恵を活用できたからこそです。AWSは知見が豊富に共有されていて素晴らしいですね。

## 19.4　コミュニケーション促進

　せっかくサービスを作るのであれば、使ってもらいたいと思うのが開発者です。今回「平成カウントダウン」を使ってもらうための工夫を用意しました。

図19.2: ツイートをコンテンツにする

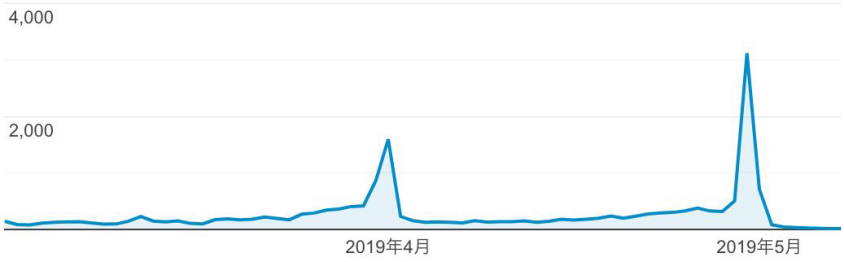

　ページ内に埋め込んだツイートボタンからツイートする際、文章の中にカウントダウンの残り時間を動的に埋め込むようにしました。そうすることでツイート自体がカウントダウンのコンテンツになるようにしています。

　ツイートの本文を動的設定する処理につまずきましたが、ボタンを押下した時に再レンダリングしてくれるメソッドがあると判明し、それを利用することで実装できました。

## 19.5　アクセスとツイート

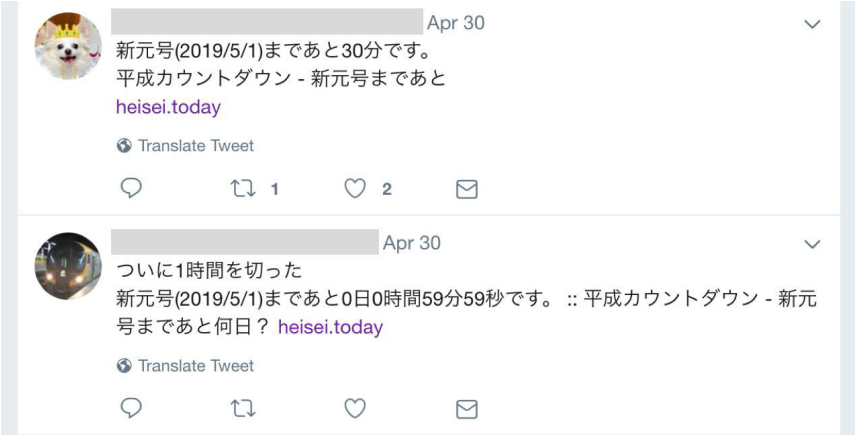
公開から2年ほど運用し、サイトへのアクセスとツイートが伸びたタイミングがふたつありました。

図19.3: 増えたアクセス




公開から1年半ほどは、日次のアクセスは100人ほどで推移していましたが、2019年4月1日の新元号発表と、2019年4月30日の新元号に改元となる直前のタイミングで、アクセスが増えました。トラフィックの大半は検索からの流入だったため、世間の関心がトラフィックに現れているのかと思います。

図19.4: 増えたツイート




前述したように、ツイートの中に新元号までの残時間を埋め込むことで、ツイート自体がコンテンツになることを狙っていました。2019年4月30日には、狙い通り、新元号へのカウントダウンツイートが急増しました。

また、2019年4月1日にツイートされた内容を見ると「4月1日に新元号になると思っていた」人が一定数いたようです。「あと1ヶ月先なの？」と混乱している様子は興味深かったです。

## 19.6　ちょっと特別な経験

　改元から約1ヶ月が経過し、サイトへのアクセスは無くなりました。サービスとしての役目を果たしたので、近いうちにクローズしようと思っています。

　もともとは、自分自身の勉強として取り組み始めました。サービスへのトラフィックから「世間の改元への注目」という社会学的な側面を垣間見ることが出来たのは、良い意味で予想外でした。元号を研究する人が、将来もしこの書籍を参考文献として引用したら熱い展開ですね！

　歴史的なイベントでこの楽しみを味わえたのは、もしかしたら人類で私ひとりだけなのかもしれません。思いつきのアイデアと少しの開発だけで、ちょっと特別な経験ができました。

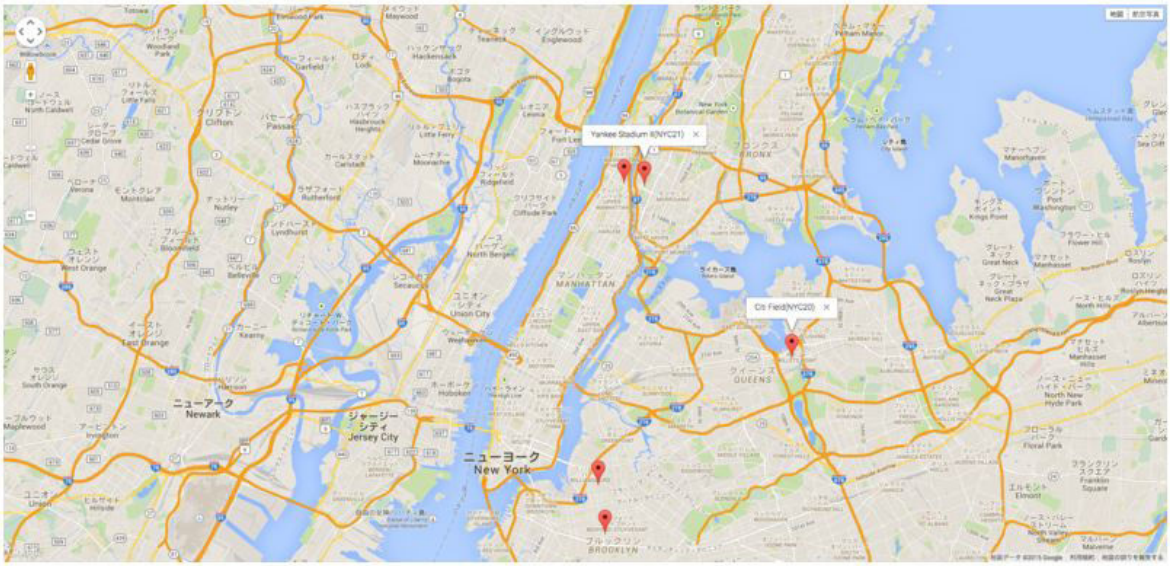
```
Takuya Beppu / @tky_bpp


WEB企業でプロダクト開発をしています。
学生時代を岡山で過ごし、ベトナムとドイツでの長期出張を経て、現在は東京。


http://tkybpp.hatenablog.com/
```

1. https://heisei.today/

# 第20章　Pythonと野球オープンデータで地図アプリを作る

　野球エンジニアの @shinyorke です。もくもく会というイベント参加をきっかけに、野球データの可視化アプリをPythonで個人開発しました。便利なデータやライブラリーを活用して、技術の勉強をしながら、自分の好きなテーマ（野球）を楽しもう！という贅沢な取り組みです。

図20.1: 地図画面



## 20.1　対象読者

以下の人にオススメします。

・Pythonでスクレイピングがしたい。

・サクッとWebアプリを作りたい。

・サーバー構築の練習をしたい。

・GoogleMapの活用やGeoCodingに興味がある。

・とにかく野球が好きだ。

コードは公開している[1]ので、どうぞ好きに持って行ってください。

野球愛溢れるエンジニアにとっては、趣味と実益を兼ねた美味しい体験になるはずです。野球に興味がないひとは、好きな分野のオープンデータに置き換えてはいかがでしょうか。サーバー構築やPythonプログラミングの勉強になるかもしれません。

## 20.2　きっかけは「もくもく会」参加

「Pythonもくもく会」[2]というイベントに参加した時に作りました。「もくもく会」は、何人かで同じ場所に集まって、各自で「もくもく」と作業や勉強をする会のことです。

イベントごとに「おしゃべりが多い・少ない」「その日にやったことを発表する・しない」の特徴があります。それぞれ一長一短あるので、自分のやることや気分に合わせて使い分けています。今回参加したイベントは、ノマド的に集中して作業や学習をするのに向いていました。

## 20.3　野球オープンデータ「Retrosheet」

Retrosheet[3]は一言で言うと「メジャーリーグの試合実況・スコアブックを元に作成された試合情報オープンデータ」です。

野球のデータといえばスコアブックです。試合の結果、打者の打席結果、投手の投球内容を、一挙手一投足まで追いかけたい。そんなデータ無いかなぁと探してたらRetrosheetを見つけました！

メジャーリーグ（以下MLB）の約80年分の試合データ（Game logs）および、各試合の打席・走塁イベント（Event file）が公開されています。

両データともcsv形式ですが、独特の構成になっています。Excelで読むことを想定しているようで、行ごとにデータ構造が違ったりと、プログラミングには向いていません。

データの解説・仕様はこちらにあります。

・Retrosheet Event files[4]

・Rstrosheet Game Logs[5]

他にも「Sean Lahman Database」[6]といった野球のオープンデータがあります。

## 20.4　野球（MLB）データベースを作る

そのままでは使いにくいデータ形式ですが、こいつをハックした猛者がいました。

・Chadwick: Software Tools for Scoring Baseball Games[7]

・wellsoliver/py-retrosheet[8]

Chadwickは、Retrosheetのデータを抽出して、意味ある単位に出力するC言語製のライブラリーです。py-retrosheetは、指定年のCSVをRetrosheetのサイトからダウンロードし、Chadwickコマンドでデータを抽出して、データベースに格納してくれるという神ライブラリーです。これらを使ってMySQL Serverにデータを入れましょう。

## 20.5　インフラ構築を自動化

　きっと同じように野球データを使いたい人はいるはずだと思い、Vagrant + Ansible で自動化しました。みんなで使えるように公開[9]しています。本書ではサンプルコードは省略します。

```
$ vagrant up
$ ansible-playbook -i hosts retrosheet_server.yml
```

　たった2行のコマンドを実行するだけで、ライブラリーのインストール、DB Scheme の作成、オープンデータのダウンロード、使いやすいようにデータの変形、DBへのデータ格納まで全て自動で行います。

　このように、勉強になりそうな技術ネタがあれば、もくもく会で試しています。学習の甲斐あって、Dockerに置き換えたデモ「hatteberg」[10]を、また別に公開しています。

## 20.6　試しにPythonでデータを使ってみる

　データ利用のサンプルコード[11]を用意しました。イチローさんが2014年、どんな打球を飛ばしたかの傾向を可視化してみます。

```
# matplotlibをimport
%matplotlib inline


from matplotlib import pyplot


# 自前で作ったライブラリーをimport
from retrosheet import RetroSheet
from database_config import CONNECTION_TEXT, ENCODING, PARAMS


# イチローのプロフィール(2014年)を取得
retro = RetroSheet(CONNECTION_TEXT, ENCODING)
player = retro.find_player_by_fullname_year("Ichiro ", "Suzuki", 2014)


# 打球のグラフを描く
# フライ(Fly Ball), つまらないポップフライ(Pop Up),
# ゴロ気味の当たり(Ground Ball), 快心の当たり(Line Drive)
labels, values = retro.batted_ball_pie_data(player.PLAYER_ID, 20140401, 20141030)
pyplot.pie(values, labels=labels)
```

図20.2: データを可視化した結果

jupyter ichirosuzuki Last Checkpoint: 2分前 (unsaved changes)

Logout

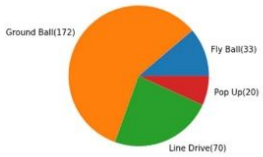
File Edit View Insert Cell Kernel Widgets Help

Trusted | Kernel

Run Code

In [1]:
```
# matplotlibをimport
%matplotlib inline

from matplotlib import pyplot

# 自前で作ったライブラリをimport ※後日公開予定
from retrosheet import RetroSheet
from database_config import CONNECTION_TEXT, ENCODING, PARAMS

# イチローのプロフィール(2014年)を取得
retro = RetroSheet(CONNECTION_TEXT, ENCODING)
player = retro.find_player_by_fullname_year("Ichiro ", "Suzuki", 2014)

# 打球のグラフを描く
# フライ(Fly Ball), つまらないポップフライ(Pop Up), ゴロ気味の当たり(Ground Ball), 快心の当たり(Line Drive)
labels, values = retro.batted_ball_pie_data(player.PLAYER_ID, 20140401, 20141030)
pyplot.pie(values, labels=labels)
```

Out[1]:
```
([<matplotlib.patches.Wedge at 0x11735a450>,
  <matplotlib.patches.Wedge at 0x1164ff090>,
  <matplotlib.patches.Wedge at 0x11735a8d0>,
  <matplotlib.patches.Wedge at 0x11735aed0>],
 [Text(1.0327686370761746, 0.3786673240085826, 'Fly Ball(33)'),
  Text(-0.9034869388608642, 0.6274642231297537, 'Ground Ball(172)'),
  Text(0.4277066026700154, -1.0134431715851036, 'Line Drive(70)'),
  Text(1.0751437287074068, -0.23252088641912175, 'Pop Up(20)')])
```




こんな結果になりました。単打が多いイチローさんらしい結果です！

## 20.7　データを使ってやりたいこと

この野球データでできること、やりたいことがたくさんありすぎます！

・投球および打席ベースの成績はほぼすべてとれるので、セイバーメトリクス[12]的な指標[13]を自分で書いたりして遊ぶ。

・ある日ある試合の打席やらスタメン表がとれるので、それを利用して超細かい野球選手名鑑（MLB）を作る。

・投球情報（しかも一球毎！）を活用して「振り返り用Pitch f/x[14]」を作る。

・球場の情報とGISを紐付けて「アメリカ野球地図」アプリを作る。これで各チーム（30球団）のオールスターとか作っちゃったり……。

今回は「地図アプリ」を作ることにしました。Geocoding（＝ジオコーディング：住所や建物名から緯度・経度を取得する技術）を学べる機会です。

## 20.8　球場データを探す

地図アプリを作ろうとして、早くも問題が発生しました。どこの球場で試合をしたかという情報が一目でわからない。試合データ（Game Logs）に球場を示すID情報はあるが、このIDに紐づく球場情報がない！どうしよう！

Retrosheetの公式サイトを調べた所、どうやら「Park Directory」ページ[15]が球場情報らしい、と分かりました。

・このサイトをスクレイピングして球場リストを作成。

・球場名をキーワードにGeocodingを実施して緯度・経度をデータとして追加。

・Databaseに突っ込んでGoogle Mapに表示。

という所までやってみよう！

## 20.9　Beautifulsoupでスクレイピング

　スクレイピングはPythonの定番ライブラリー「Beautifulsoup」[16]を使いました。書いたコード[17]はこんな感じです。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

__author__ = 'Shinichi Nakagawa'

import json
import urllib.request
from bs4 import BeautifulSoup
from configparser import ConfigParser
from geopy.geocoders import Nominatim



class BallParks(object):

    def __init__(self):
        config = ConfigParser()
        config.read("config.ini")
        html = urllib.request.urlopen(config["BallParks"]["url"])
        self.soup = BeautifulSoup(html)
        self.geolocator = Nominatim()

    def _geocode(self, name):
        """
        Geocoding
        :param name: park name
        :return: lon, lat
        """
        location = self.geolocator.geocode(name, timeout=600)
        if location is not None:
```

```
            return location.longitude, location.latitude
        else:
            return 0.0, 0.0


    def _get_parkid(self, href):
        """
        URLからparkidを取得
        :param href: url text
        :return: parkid
        """
        # urlを"_"で分けて後ろの方をGet
        params = href.split("_")
        return params[1].replace(".htm", "")


    def list(self):
        """
        球場リストをGet
        :return:
        """
        parklist = []
        for a in self.soup.find_all("a"):
            href = a.get("href")
            # 「PK_」を含むURLがある場合は球場を示す（らしい）
            if "PK_" in href:
                print(a.text)
                lon, lat = self._geocode(a.text)
                parklist.append(
                    {
                        "name": a.text,
                        "url": href,
                        "parkid": self._get_parkid(href),
                        "lon": lon,
                        "lat": lat,
                    }
                )
        return parklist
```

```
if __name__ == '__main__':
    ba = BallParks()
    with open('parklist.json', 'w') as outfile:
        json.dump(ba.list(), outfile, indent=4, sort_keys=True)
```

## 20.10　geopyでらくらくGeocoding

球場の名前と（謎の）IDは楽に取れましたが、Geocodingについては悩みました。住所や建物名から緯度・経度を取得したい。

適当にググった所、いい感じのライブラリ「geopy」[18]を発見しました。GoogleやBing、OpenStreet Map（OSM）といった地図プロバイダーのAPIを使いやすくしたPythonライブラリーです。

```
>>> from geopy.geocoders import Nominatim
>>> geolocator = Nominatim()
>>> geolocator.geocode("Tokyo Dome")
Location(東京ドーム (Tokyo Dome), 牛込小石川線, 飯田橋, 東京都,
関東地方, 112-8555, 日本, (35.70556965, 139.751887568794, 0.0))
```

これだけのコードでGeocodingができちゃいます。たった3行で欲しいデータが取れちゃうなんて便利です。

今回は「Nominatim」というOSMのGeocoding APIを使いました。スクレイピングした球場名を元に緯度経度をゲットしました。

## 20.11　bottle + Google Map APIでサクッっと地図アプリを作る

　このデータをDatabaseに入れましょう。早く地図を見たかったのでMySQLは一度忘れて、気軽に作れるSQLite3にデータを入れました。create_database.py [19]というコードでGitHubに公開しています。

　Google Map APIを使って地図を可視化するWebアプリを作りました。「bottle」という軽量Web Frameworkで実装しています。公式ドキュメント[20]にHello Worldを表示するためのお手軽手順が載っています。

　アプリ本体はこんな感じ[21]で書きました。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

__author__ = 'Shinichi Nakagawa'

from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from tables import Park, Base

from bottle import route, run, template
from configparser import ConfigParser

config = ConfigParser()
config.read("config.ini")



MAP_API_KEY=config["google"]["api_key"]
MAP_CENTER = {
    'lat': 37.751526,
    'lng': -122.200470,
}



def get_session():
```

```
    engine = create_engine('sqlite:///ballpark.db', echo=True)
    Base.metadata.create_all(engine)
    Session = sessionmaker(bind=engine)
    return Session()



@route('/')
def index():
    session = get_session()
    return template(
        'map',
        map_key=MAP_API_KEY,
        sensor='false',
        center_lat=MAP_CENTER['lat'],
        center_lon=MAP_CENTER['lng'],
        parks=list(session.query(Park).all())
    )



if __name__ == '__main__':
    run(host='localhost', port=8000, debug=True, reloader=True)
```

HTMLテンプレート views/map.tpl [22]は以下のように書きました。

```
<!DOCTYPE html>
<html>
  <head>
    <title>Baseball Park Map</title>
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no">
    <meta charset="utf-8">
    <style>
      html, body, #map-canvas {
        height: 100%;
        margin: 0px;
        padding: 0px
```

```
      }
    </style>
    <script src="https://maps.googleapis.com/maps/api/js?v=3.exp&key={{ map_key }}&sensor={{
sensor }}">
    </script>
    <script>
        var map;
        function attachMessage(marker, msg) {
            google.maps.event.addListener(
                    marker, 'click',
                    function(event) {
                        new google.maps.InfoWindow(
                                {
                                    content: msg
                                }
                        ).open(marker.getMap(), marker);
                    }
            );
        }
        function initialize() {
            var parks = new Array();
        % for i, park in enumerate(parks):
            parks.push(
                    {
                        position: new google.maps.LatLng({{ park.lat }}, {{ park.lon }}),
                        content: '{{ park.name }}({{ park.parkid }})'
                    }
            );
        % end
            var mapOptions = {
                zoom: 3,
                center: new google.maps.LatLng({{ center_lat }}, {{ center_lon }})
            };
            map = new google.maps.Map(
                    document.getElementById('map-canvas'),mapOptions
            );
```

```
            for (i = 0; i < parks.length; i++) {
                var myMarker = new google.maps.Marker(
                        {
                            position: parks[i].position,
                            map: map
                        }
                );
                attachMessage(myMarker, parks[i].content);
            }
        }
        google.maps.event.addDomListener(window, 'load', initialize);
    </script>
  </head>
  <body>
    <div id="map-canvas"></div>
  </body>
</html>
```

これでDatabaseの中身を見る地図アプリが完成しました。

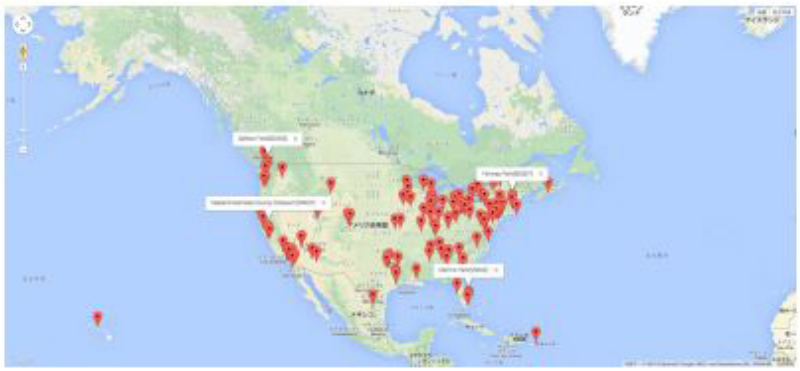
## 20.12　地図アプリの完成

地図アプリを実行します。

```
$ python map.py
```

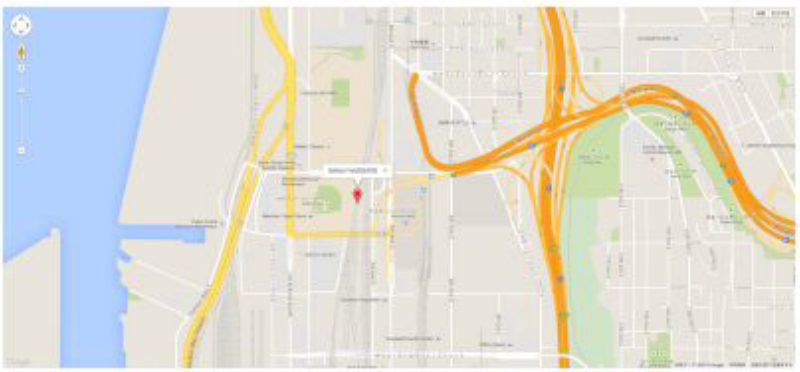
localhostにアクセスすると、無事に地図が表示されました。

図20.3: 地図



大雑把な位置は合っているようです。

図20.4: 地図



セーフコ・フィールド（マリナーズ本拠地）は多少ズレていますが、まあこれぐらいなら許容範囲でしょう。Geocoding APIを変えたら解決する可能性があります。

## 20.13　まとめ

もくもく会の参加をきっかけに、野球オープンデータ「Retrosheet」とプログラミング言語「Python」（bottle、Beatutifulsoup、geopy）を使って、地図アプリを個人開発してみました。チャンスやツールはいくらでも転がっています。本気で探せばきっと手に入るはずです。ぜひ興味のあるテーマを見つけて楽しみましょう！[23]

```
Shinichi Nakagawa / @shinyorke

「セイバーメトリクス」と呼ばれる野球統計が大好きな「野球エンジニア」。
「Pythonもくもく自習室」という勉強会をほぼ月イチで主催しています。

https://shinyorke.hatenablog.com/
```

1. https://github.com/Shinichi-Nakagawa/retrosheet_ballpark_database
2. https://mokupy.connpass.com/event/13840/
3. https://www.retrosheet.org/
4. https://www.retrosheet.org/eventfile.htm
5. https://www.retrosheet.org/gamelogs/index.html
6. http://seanlahman.com/baseball-archive/statistics/
7. http://chadwick.sourceforge.net/doc/index.html
8. https://github.com/wellsoliver/py-retrosheet
9. https://github.com/Shinichi-Nakagawa/retrosheet-mysql-server
10. https://github.com/Shinichi-Nakagawa/hatteberg
11. https://github.com/Shinichi-Nakagawa/retrosheet-app-example
12. 野球におけるデータ（選手成績、試合の結果、球場のスペックetc...）を統計学的に分析を行い、選手の能力、チームの強さなどといった事を分析、チームの経営や戦略に役立てる手法や考え方のこと。
13. 出塁率、OPS（出塁率+長打率）、QS（クオリティ・スタート）、ピタゴラス勝率など。
14. 球速等を計測する高性能スピードガン
15. https://www.retrosheet.org/boxesetc/MISC/PKDIR.htm
16. http://kondou.com/BS4/
17. https://gist.github.com/Shinichi-Nakagawa/4e0d673a786cc8f01f09
18. https://github.com/geopy/geopy
19. https://github.com/Shinichi-Nakagawa/retrosheet_ballpark_database/blob/master/create_database.py
20. http://bottlepy.org/docs/dev/index.html
21. https://gist.github.com/Shinichi-Nakagawa/c5cb15f34a14f67381d3
22. https://gist.github.com/Shinichi-Nakagawa/adcc680e7c8b02a54b0d

23. 本稿は執筆者（@shinyorke）によるブログ記事『【野球】30分でわかるセイバーメトリクス』『最強の野球オープンデータ「Retrosheet」をPythonでHackしてゲームに勝つる何かを作ろう（序章）』『最強の野球オープンデータ「Retrosheet」をPython+Vagrant+Ansibleで誰でも使えるようにしました』『【Python】bottle,Beautifulsoup,geopyを使って野球の地図を作ってみました』『【野球Hack】PythonとJupyterで「一球速報」っぽいモノを作る(MLB編)』を加筆・修正して寄稿したものとなります。

# 第21章　困りごとを1日で解決する「どやっ」駆動のツール開発

　ソフトウェアエンジニアの@shoitoです。Chrome拡張を始めとして30本近くのツールを個人開発しています。作ったツールの紹介と、周りに「どやっ」と自慢できるような開発スタイルをお伝えします。

## 21.1　どんなツールを作ってるか

Chrome拡張、astahというソフトウェア設計ツールのプラグイン、最近はSlack関連（Bot, Slash Command, App）が多いです。

・Chrome拡張: 11本
・プラグイン: 6本
・Slack関連（Bot, Slash Command, App）: 4本
・ライブラリー（JavascriptやJava）: 4本
・Webサービス: 3本

Chrome拡張が多いのは、プライベートや業務で使っているWebサービスに、独自の改善機能を追加できるからです。今回は「勤怠管理」のChrome拡張をご紹介します。

MyレコーダーChromeアシスタント

勤怠管理システム「Myレコーダー | KING OF TIME」を快適に使えるようにするためのChrome拡張[1]です。

図21.1: MyレコーダーChromeアシスタント


この勤怠管理システムは「出勤」「退勤」のボタンを押すだけで出退勤記録ができます。しかし、会社のSlackでは「打刻を忘れた。Slackと連携して欲しい。」「勤怠の押し忘れが多すぎる。この問題は早く改善して欲しい。」といった声がポツリポツリと挙がっていました。

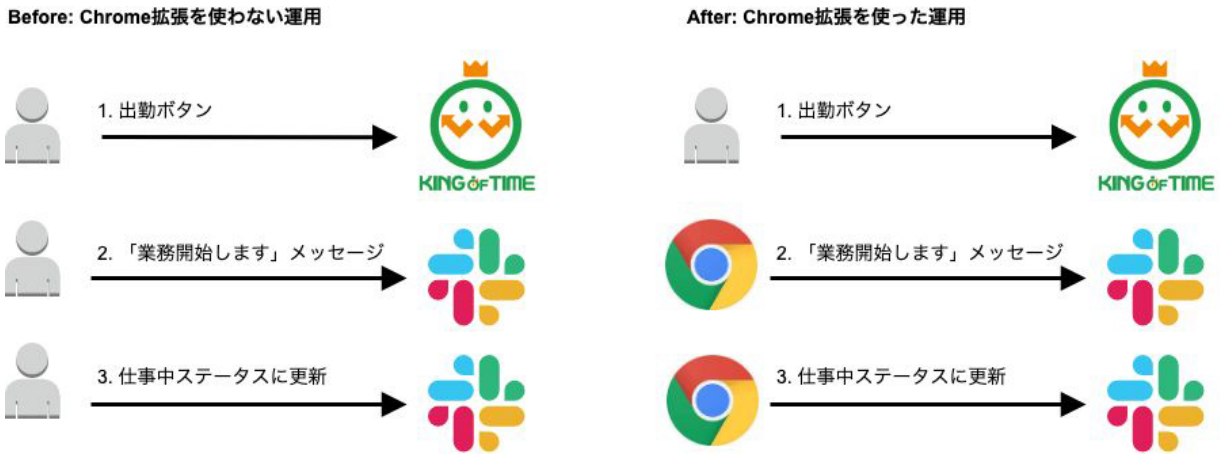
入力漏れが多い原因は何か。出退勤時のアクションが多いからではないか。そう考えました。「Myレコーダー | KING OF TIME」とSlackとにツールが分散し、その両方で以下のアクションが必要です。

1．毎日出退勤時に「Myレコーダー | KING OF TIME」から申請する。会社として勤怠状況を管理するためです。

1．リモートワークや別拠点でも稼働が分かるように、Slackのメッセージで業務や休憩の開始・終了を伝える。同僚に勤怠状況を伝えるためです。

これらをより少ない操作で済ませるために、Chrome拡張を開発しました。Chromeのアクションボタンからポップアップ表示できるようにし、手軽に「出勤」「退勤」を申請できます。さらに2のアクション（Slack投稿）をChrome拡張が自動で実行します。

図21.2: 勤怠管理のBefore, After



## 21.2　個人開発ソフトウェアを一挙紹介

他には以下のようなツールを作りました。

Chrome拡張: 11本

1．MyレコーダーChromeアシスタント：勤怠管理システム「Myレコーダー | KING OF TIME」とSlackをコラボして便利にする拡張。
2．Developers.IO編集者アシスタント：クラスメソッド発「やってみた」系技術メディアのブログ記事編集を便利にする拡張。
3．Typetalk Notifications：ビジネスチャットツールTypetalkの通知の未読数表示する拡張。
4．nicomnibox：Chromeのアドレスバーからサジェスト機能に導かれながらniconicoのコンテンツを簡単に検索できる拡張。
5．DataURI2Image：Data URIスキームがどんな画像なのか、のような文字列から画像を確認できる拡張。
6．それ、どんな本？：Amazonや楽天ブックスなどで今見ている本が、どんな本か要約などを教えてくれる拡張。
7．ChontentEditable：今開いているページをリアルタイムに自由に編集可能にする拡張。
8．java2yuml：GitHubでJavaコードを読んだり、JavaDocを読んでいるときに、クラス図を表示して理解を助けてくれる拡張。
9．Cacoo Finder：Cacooの図やシート、コメントを確認したり、簡単にコメントが付けられる拡張。
10．ReverseIt：Chromeで開いているWebページを左右反転させるネタ拡張。
11．Microjs CodeReading Bookmarklet：JavaScriptライブラリーのソースコードリーディングを支援するブックマークレット。

プラグイン: 6本

1．Cacoo Diagramming for Confluence：Caccoで描いた図をConfluenceのWikiページに埋め込めるConfluenceプラグイン。（株）ヌーラボへ移管され公式提供されたため、お役目御免。
2．Backlog Connector for astah：Backlogの課題情報をastahのマインドマップにインポートするastahプラグイン。
3．FreeMind Import for astah：FreeMindのマインドマップをastahのマインドマップにインポートするastahプラグイン。

4．Cacoo Import for astah：Cacooで描いた図をastahの設計情報としてインポートするastahプラグイン。
5．astah Nannyプラグイン：長時間使用しているとクラウディア・窓辺（マイクロソフトの公認キャラクター）が休憩するように優しく促してくれるastahプラグイン。
6．astah Tipsプラグイン：Tips情報を表示するastahプラグイン。公式に「スタートアップ画面」が提供されお役目御免。

Slack関連（Bot、Slash Command、App）: 4本

1．日本語 <---> ベトナム語通訳ボット：ベトナムチームとのオフショア開発をフォローするためのSlackボット。
2．Slackチャンネル検索コマンド：Slackのチャンネル名、目的、トピックでチャンネルを検索するSlashコマンド。2019年3月にSlackが公式で機能提供を開始したため、お役目御免。
3．Slackまとめて招待コマンド：Slackでチャンネルから他のチャンネルへ参加メンバーをまとめて招待するSlashコマンド。プライベートチャンネルには対応できない。/whoコマンド で代用できたため、お役目御免。
4．Slack KOT Bridge：SlackメッセージでKING OF TIMEの出退勤処理ができるSlackアプリ。

ライブラリー(JavascriptやJava): 4本

1．search-nico-js：niconicoコンテンツ/タグ検索APIのJSライブラリー。APIのバージョンアップによりお役目御免。
2．typetalk-js：ビジネスチャットツールTypetalkのAPI用JSライブラリー。
3．as3webstorage - Web Storage API wrapper for ActionScript：ActionScriptからJSのWeb Storage APIを使えるラッパーライブラリー。
4．as3geolocation - Geolocation API wrapper for ActionScript：ActionScriptからJSのGeolocation APIを使えるラッパーライブラリー。

Webサービス: 3本

1．japarser：Javaソースコード解析Web API。抽象構文木（AST）からクラス情報をJSON形式で返すAPI。
2．niconicoタグ回遊：niconicoコンテンツ検索APIのデモWebアプリ。タグでコンテンツを回遊できる。
3．美人百景：美しい女性、綺麗な女性、可愛い女性をいつでも堂々と好きなだけ眺められるWebアプリ。

## 21.3 「えいや！」→「どやっ！」→「いいね！」の成功体験

私の個人開発を振り返ると、自分や身近な人の課題解決をするツールを作ることが多いです。「いいね！」「すごい！」「ありがとう！」などのフィードバックが成功体験になって、開発を続けられているのだと思います。

SlackやTwitter、日常の会話の中で「xxxが面倒くさい」「xxxできるともっとイイ」のようなメッセージを見つけます。共感できる課題があれば、頭の中でざっくりと設計してみます。実現できそうだと判断したら、短期間で「えいや！」と作ります。それを「どやっ！」と提供して、フィードバックを楽しみます。

「短期間」と書きましたが、開発期間は数時間から2週間程度です。課題に気付いてからリリースまでの期間が短ければ短いほど「どやっ」としたときに驚いてもらえます。大抵は「翌日まで」を目標にして、あとは勢いで作ります。

## 21.4　気をつけているポイント

仕事ではないので、軽い動機で始めることが多いです。好きなサービスやAPI、プラットフォームを使って何か作ってみたい。その程度でイイと思っています。

その一方で、私がツール開発をする上で気をつけているポイントが3つあります。

１．シンプルさ

２．コントロール可能な配布の仕組み

３．自分がユーザーになれる

それぞれのポイントを解説します。

### 1：シンプルさ

一つの目的に特化した、分かりやすいものだけを作ります。シンプルであるため、開発中に方針もブレにくいですし、開発期間も短く済みます。

逆に、複雑化してしまうと、リリースまでの時間が長くなってしまいます。私は飽きっぽい性格なので、開発熱が冷めてしまいます。

### 2：コントロール可能な配布の仕組み

ツールをユーザーに周知し、インストール＆アップデートし続けてもらうのは難しいです。また、どれくらいのユーザー数か、どんな属性の人たちか、トラッキングする仕組みまで自分で準備するのは無理があります。

そこで、AppStoreやGoogle Playストア、Chromeウェブストア、Firefox Add-onsのような、プラットフォームを活用します。公開URLが発行されるので、ユーザーに簡単に配布できます。新バージョンのリリース通知や自動アップデートの仕組みがあるため、気楽にバグ修正や機能追加ができます。

開発者としてはユーザーに最新版だけを使ってもらいたいものです。複数のリリースバージョンのメンテナンスをするのは辛いので。

### 3：自分がユーザーになれる

自分がユーザーであればドッグフーディングをしながら開発ができます。ツールを作って楽しみ、ツールを使って楽しむ、そんな一石二鳥の体験です。

せっかくの個人開発なら、贅沢に楽しみたいですよね。もし誰も使ってくれなくとも、少なくとも自分がユーザーで、自分にとっての価値を提供できれば、開発のモチベーションも続きます。

さいごに

　個人開発は自由です。自分がプロダクトオーナーであり、デザイナーであり、開発者であり、マーケティング担当者であり、サポート担当者です。もちろん苦手なところは協力を仰いでも良いと思います。

　自分が良いと信じたものを、楽しみながら開発して、ユーザー（自分だけだってかまわない！）にも喜んでもらえたら、テンションが上がると思いませんか。

```
Sho Ito / @shoito


Elasticsearch,Java,Golang,JavaScriptに触れるソフトウェアエンジニア。
ソフトウェア設計支援ツールや検索サービス基盤の企画・開発・運用を経験。


https://github.com/shoito
```

1. https://chrome.google.com/webstore/detail/pifbdpooppfkllaiobkaoeocbfmpabaj

# 第22章　10年つづくWEBサービスの育て方 ～地道な宣伝から法人契約まで～

こんにちは、もぎゃと申します。フリーランスのWEBエンジニアです。

街の電源検索サイト「モバイラーズオアシス」を運営しています。ノートパソコンやスマートフォンのために、電源コンセントを使えるカフェや図書館、待合室などの情報を地図上で探すことができます。2009年に公開してから、もう10年以上運営しています。ここでは細々とでもサービスを10年続けることで得られた知見を提供しようと思います。

## 22.1　街の電源検索サイト「モバイラーズオアシス」

　コーディング作業で煮詰まってしまい場所を変えたい時や、外出先でバッテリーがなくなりそうな時に、電源コンセントが提供されているお店を探し回った経験はありませんか。そこで活躍するのが「モバイラーズオアシス」[1]です。

図22.1: モバイラーズオアシス


　ここ10年でノートパソコンやスマートフォンが急速に普及しました。「バッテリーはあとどのくらい持つだろうか」と多くの人が充電を気にしています。

　その中にはプログラミングができる人もいます。何人かは「よし、充電スポットの検索サイトを作ろう！」と思い立ちます。そんなわけで、年に一つずつくらい新しいサイトが登場する、新規参入の多い分野だったりします。

　けれども、多くのサイトはやがて更新されなくなり、朽ちていきます。10年間「モバイラーズオアシス」を続けているからわかるのですが、電源検索サイトはそんなに儲からないのです。

## 22.2　サービス開発のきっかけ

筆者はお客様と打ち合わせをするとき、早めに現地に到着し近隣のカフェで仕事をします。この時に選ぶカフェは「電源と無線LANがあって、混雑していなくて、長居しても迷惑にならない」という条件が揃っていると理想です。何度も訪れた街であれば、どのカフェが条件を満たしているのか、だんだんわかってきます。しかし、行き慣れない街だと、条件にあった場所を見つけるまで、いくつものお店を試す必要があります。

「〇〇駅 電源 カフェ」とキーワード検索すれば、ブログのまとめ記事を見つけることができます。しかし、ブログ記事はあまり更新されません。古い記事を参考にすると、電源が使えなくなっていたり、そもそもお店自体が存在しなくなっていたりします。

そういったサイトを何度も見ているうちに「もしかしてみんな困っているのではないか」「充電できる場所を検索できるサイトを作ろう」と思い立ちました。ブログに散らばっている情報を探すよりも、電源が使える場所が地図に掲載されていれば、目的地周辺の電源が使えるお店を視覚的に探すことができます。情報が一箇所に集まっているサイトなら、登録した人がその後行かなくなったとしても、次に行った人が情報を更新できます。きっとみんなの役に立つ！

筆者はフリーランスのWEBエンジニアです。WEBサイトの開発経験がありました。加えてこの前の年、サラリーマン時代に購入したマンションを売却して、数年間は働かなくても食べていけるだけの現金も持っていました。個人開発者は、時間の捻出に悩んだり、家族の理解を得ることに苦労します。筆者の場合、そういった問題に悩むことなく、じっくりと開発することができたのはとてもラッキーでした。

### デザインをコンペで募集するが……

WEBサイトの開発は慣れていたのですが、デザインスキルには自信がありませんでした。そこで、そのころ有名になり始めていたランサーズでコンペを開催しました。

「電源のあるお店を検索できるサイトを開発しています。こんな感じで動いてるのですけど、ちゃんとしたデザインが欲しいのでページデザインをお願いします。PhotoShop/Illustlatorで作成してくれたら、HTML/CSS化はぼくがやります」

賞金5万円のコンペで、9人のデザイナーに応募いただきました。このデザインです。

図22.2: 提案していただいたデザイン


今はロゴしか残っていません。コンペ募集の時点では、検索フォームに「場所：新宿駅、カテゴリ：スターバックスか銀座ルノアール、無線LAN：ソフトバンクWi-Fiか登録なしで使えるフリーWi-Fi」と条件を入力して、検索結果を返すサイトを想定しました。

けれども、しばらく運用した結果、新宿駅などの極端な例を除き、1つの駅に電源が使えるお店はせいぜい10件。あれこれ条件を入力せずとも、指定した駅周辺のカフェを全部表示すれば十分でした。現在では、場所だけ入力したら、周辺にあるよさそうなお店を全部表示して、ユーザーに判断してもらっています。

マクドナルドやスターバックスといったチェーン店を絞り込むための一覧デザインや、位置情報を指定するフォームなど、デザイナーにたくさん提案いただいたのですが、そもそもの依頼事項が間違っていたので、ほとんど無駄になってしまいました。

教訓：デザインの依頼は、一度公開して、具体的な反響や課題が得られてからにしたほうがいい

## 22.3　サービス公開も反応なし、地道なデータ入力

筆者は会社員時代、友人たちと一緒に「いいめも」というサービスを公開したことがあります。携帯メールでおこづかい帳をつけるサービスです。優秀な友人のマーケティング能力に加え、各種メディアにつながりを持つ方からのバックアップもあって、それなりの反響をいただきました。筆者が会社をやめてフリーランスになったきっかけでもあります。このときの経験から、筆者は勘違いをしました。「公開しました」とツイートしたら、みんな広めてくれて、どっかんどっかん反響があると思っていたのです……。

そんな都合のいい話は何度もありませんでした。モバイラーズオアシスのリリースは、友人が一言コメントをくれただけで、ほぼ何の反響もなく静かに終わりました。

図22.3: 誰一人反応してくれなかった


「愛情の反対は無関心」と言いますが、なるほどこういう意味なのだなと痛感しました。自分が欲しくて作ったツールですし、続けていればそのうち反響もあるだろうと思い、しばらくは地道な活動を続けることにしました。

最初は、ブログサイトや他の類似サイトに掲載されている情報を、ひとつひとつ手作業で入力しました。手作業では効率が悪いと気づいたので、以前仕事のために開発した「Excelデータをデータベースに流し込むスクリプト」を利用して、いったんExcelにまとめたデータをモバイラーズオアシスに流し込むようになりました。

しばらく経つと、それぞれのサイトに掲載されているデータには、決められたフォーマットがあることに気がつきます。スクレイパーという、人間の代わりにサイトを閲覧して情報をとってくるプログラムを書くようになりました。これでCSVファイルを出力すれば、モバイラーズオアシスに流し込むことができます。

やがてこのソフトは進化を繰り返し、CSVを経由しなくてもデータベースに直接接続してデータを更新するようになりました。チェーン店の公式サイトから店舗情報を取得するようになり、現在では数十のロボットが夜中にあちこちのサイトを巡回するようになっています。このプログラムのおかげで、新しく開店したお店や営業時間の変更などが自動的に反映されるようになりました。

### Twitterでの宣伝活動、だんだんとサイトが軌道に乗る

フォーマットの決まったサイトはロボットに任せることができたので、今度はブログやTwitterのように自由形式で書かれた情報の収集に注力し始めました。「電源 カフェ」でTwitterを検索すると「ここのカフェは電源が使えた。嬉しい！」みたいなツイートが見つかります。これをサイトに掲載するということを繰り返しました。

残念ながら「喫茶室ルノアール 新宿南口ルミネ前店：電源が使えました」というような都合のいいツイートはほぼありません。「新宿のマクド電源使えたマジ神」くらいの曖昧な"つぶやき"がほとんどです。そこで、投稿者に「こんにちは。電源が使えるお店の情報を集めているモバイラーズオアシスというサイトです。先程のマクドナルド、このお店であってますか？」とお問い合わせをするようにしました。お返事をいただけたらすぐに情報を掲載して「ありがとうございます！おかげでまた一つ、電源が使える場所の情報を増やすことができました！」というリンク付きのツイートを送ります。質問に答えてくれた人にモバイラーズオアシスを知っていただこうという作戦です。

図22.4: 「あ、自分のツイートが役に立ったんだ」と思えば悪い気はしないですよね


電源について検索すると「新宿で電源の使えるカフェどこ？」とツイートしている人も見つかります。そこで個別に「こんにちは。モバイラーズオアシスというサイトでこんな地図で見ることができます。よかったら試してみてね！」と宣伝ツイートを送りました。

見ず知らずのユーザーに対して「サイトを使ってみてね」と呼びかける。わりと無茶な宣伝方法でしたが、リアルタイムで需要に応える活動だったおかげか、意外と怒られることはなく、地味にユーザーを増やせました。なかには「こんな情報が来て便利だったよ」と宣伝に協力してくれることもありました。

こうして地道な宣伝を繰り返していると、同じように情報を集めている人を見つけることができます。そのうち何人かは、定期的にモバイラーズオアシスに情報を提供してくれるようになりました。1年が経った頃から、こちらからお願いしなくてもツイートや情報提供フォーム（掲示板）から情報を寄せてくださる方が現れるようになりました。また、佐々木俊尚さん、けんすうさんなどの有名人に取り上げてもらえたり、ブログ記事で書いたネタがバズるなど、徐々に手応えが得られるようになってきました。

スマホアプリの開発をクラウドソーシングで依頼

モバイラーズオアシスを開始した2009年は、スマホアプリの普及期です。競合サービスの中にはアプリでデータ提供を開始するところが出始めました。

モバイラーズオアシスのアプリ版も作ろうと思い、クラウドソーシングを利用しました。良いデザイン提案を頂いて気に入っていたので、ランサーズでお願いしました。

何人か応募を頂き、大学生だけどアルバイトでアプリ作っています！というAさんに依頼しました。金額は30万円。立ち上げ期の個人サイトとしては思い切った投資でした。

筆者「データの提供形式はこんな感じなのだけれど」

Aさん「じゃあ2週間位でできると思います！」

筆者「（頼もしいなぁ）」

～2週間後～

筆者「（あれ？何も連絡がないな……） Aさん、その後どうなってますか？」

Aさん「すいません。もう少し掛かりそうです」

筆者「（これは危ないパターン）とりあえずあと1週間待ちます。その時点で出来たものは全部提出してください」

～1週間後～

Aさん「すいませんできませんでした。ソースコードです」

筆者「（ふむふむ。これは、なんにもしてなかったわけじゃないけれど、提供したデータをアプリ用に変換するあたりで時間を費やしてしまったパターン……。相談してくれたら、使いやすく変換した形で提供したのに……）」

Aさんと対策を話し合って、結局、2ヶ月後にアプリをリリースできました。ところが、WEBサイトの更新とともに、アプリにも機能を追加したくなります。すでに30万円のお金を使っちゃったのに、この先機能を追加するたびに開発費を払うの！？かといって他人の書いたコードをメンテナンスできるほど筆者のアプリ開発力は高くないので、結局このアプリは、1年程度そのまま放置することになってしまいました。

教訓： アプリを作る時は、その後機能追加することを前提に考えておかないといけない

スマホアプリの開発（続）

この数年後、ガソリン価格を地図で見ることができるアプリを開発していたPさんと知り合って、電源検索アプリを作る話が再び持ち上がりました。前回の反省を生かして、アプリに掲載した広告費の半分をお渡しするというレベニューシェア契約にしました。

Pさんはアプリの開発実績があり、筆者も二回目の開発委託だったので、話は順調に進み、モバイラーズオアシスのアプリ第二弾がリリースされました。筆者自身はメンテナンスすることなくアプリを提供できて、いい話だと思っていたのですが。

「Pさん、今月の広告費が◯円だったので、半分の◯円をお振込しました」

「Pさん、今月の広告費が◯円だったので、半分の◯円をお振込しました」

「Pさん、今月の広告費が◯円だったので……」

複数の広告を別の会社から配信しているので、別々のサイトから売上を拾ってきて、その半分を計算して振り込んで……。めんどくせえぇぇぇぇぇ！！！！！

売上はどうせ管理するのだから、どうってことないような気もするのですが、Excelシートを起こして、計算書を作って、毎月振込み処理を行って、メールを書く、その手間が、その時の筆者には、とても面倒でした。1人で売上を総取りするのなら、年に1回まとめて帳簿に転記すれば済むのに、レベニューシェアのためにこの作業を毎月やらないといけない……。

結局、この契約は、およそ1年で、筆者からお願いして終了にしていただきました。会社規模でサービスを運営して、経理担当者がいるならば、どうということのない作業だったと思います。1人で開発も経理も総務もこなす筆者には、この手間は耐え難いものだったのです。

その後、筆者は真面目にアプリ開発を勉強し、最終的には自分でアプリをリリースしました。とはいえWEBサイトに比べるとメンテナンスに手が回っておらず、1人で開発する限界を感じているのが現状です。手が回らないアプリには酷評がついてしまうし、これだったら最初からWEBに集中していたほうが良かったかもしれません。

教訓：WEBサイトとアプリ開発を両立するのは難しい

## 22.4　サイトで扱っているデータを法人に販売

アプリ開発が迷走する一方で、WEBサイトは月に約80,000PV、1万円の広告収入と順調に成長をしていました。お小遣いとしては悪くない額です。ただ、モバイラーズオアシスはロボットが常に稼働して情報収集しているので、普通のサイトよりもサーバー代がかかります。マンションを売ったお金にも限りがあるので、もう少し売上が出ないとサイトのメンテナンスは難しくなります。

有料プランを作ってヘビーユーザーの方々からお金をいただくことも考えたのですが、結局はやめました。モバイラーズオアシスを気に入って使ってくれている方々は、新しいお店の情報を提供してくれたり、データが間違っていたら教えてくれたりする、重要な情報源でもあります。この人たちからお金をもらうのは抵抗感があり、ヘビーユーザーの方々が離れてしまうのも怖かったので、別の収益源を探しました。

そこで目をつけたのが地図会社です。マピオンをはじめとした地図会社では、付加価値としてエステマップやカレーマップなど、カテゴリ別に便利な情報をまとめてユーザーに提供しています。こういったサイトであれば、モバイラーズオアシスが持っている「電源が使えるお店の情報」にお金を出してくれるのではないかと考えたのです。

ジオメディアサミットという地図に関するサービスの関係者が一堂に会するイベントがあります。この懇親会で名刺交換をして、良さそうな会社の人に話しかけました。

「モバイラーズオアシスという電源検索サイトを運営しています。ここに載ってるお店の情報を提供してお金をもらいたいのですが、いかがでしょうか？」

最初は「駄目って言われるだろうからどの辺りが断る理由なのか教えてもらおう」くらいの気持ちでした。ですが、意外とどの会社も前向きに持ち帰って、社内で検討してくれました。

あとで知ったことなのですが、地図会社がこのようにデータを購入するのは一般的なことで、このことに特化した事業会社さえ存在するそうです。全国の主婦をアルバイトで雇用して、駅の出口に近い電車の号数を調べたり、駅のトイレの様子をまとめて地図会社に販売する。そういった専門の会社に比べると、モバイラーズオアシスの情報はずっと安価で、他社が手がけていない分野だったので、検討する価値があったようです。

結果的に駄目だった会社では、情報の信頼性が問題になりました。WEB上で誰ともわからない人が言ってるその情報、本当に正しいの？ということです。地図の会社は信頼性が命なので、そのレベルには達していないと判断されてしまったようです。一方で、たしかに甘いけれども、同類のサイトよりはずっと精度が高いし、この金額だったらアリじゃね？と思ってくれた会社もあって、めでたく契約にこぎつけることができました。

司法書士に契約書の内容を相談

個人サイトが法人からお金をいただくに当たって、契約書をどうするんだというのが問題になります。法人にはたいてい顧問弁護士のような方がいるので、先方の作った契約書にただ印鑑を押してしまえば楽なのですが、その結果、不利な条項が書かれていて、タダのような値段でサービスをまるごと売却したことになっていたり、意図しない作業を強いられることになったら目も当てられません。筆者の場合、お渡ししたデータで類似サイトを立ち上げられるのは困りますし、お渡ししたデータを倍額で他社に転売されるのも避けたいところです。ちゃんとした契約を検討する必要がありました。

「契約書作成 司法書士」とググったところ、近所で開業したばかりの若手司法書士が見つかったので、メールしてみました。会ってみると、年齢が近いこともあり気が合って、数万円程度の報酬で契約を手伝っていただけることになりました。

しかしながら、契約書の作成というのは意外とタフな交渉です。最初の草案は先方に作ってもらったのですが、それを司法書士に見ていただいて、その内容をもとに先方に修正を提案すると、そのとおりに修正されることはなくて、「これでいかがでしょう？」という別案が出てきます。その内容を司法書士に見ていただいて、修正を提案して、別案を司法書士に見ていただいて……。何回かやり取りをしたところで、司法書士の方がイラっとし始めたのを感じ取りました。毎回ちゃんと調査してアドバイスをしてくれているので、メール1回やり取りするだけでもそれなりの時間がかかります。どう見たって採算の合わない案件です。悪いなあとは思うものの、筆者もはじめての取引で油断するわけには行かないので、この時はそのまま押し切って最後までお付き合いいただきました。

そして次の取引からは、「先方から示された契約書を見て、問題点や懸念点を指摘していただく。その後の契約修正については関与しない」という契約にしました。

契約書をめぐって「こんなことになるなんて……！」となるケースは、だいたい司法書士に相談せずに、先方から示された契約書にそのまま印鑑を押してしまって発生します。問題になる条文は最初から契約書に埋め込まれているのだから、最初に1回チェックしてもらえば、危険なところは見つけてもらえるはずです。その後の交渉で書き換えた部分に罠を埋め込まれる可能性もあるのですが、お互い交渉して注目している箇所に罠を埋め込むのは簡単じゃないし、そこは自分が注意することでコストとリスクのバランスをとろう、という目論見です。高額の契約なら弁護士にお願いして全部手伝ってもらうのもいいですが、そこまで出せない場合、最初1回だけ司法書士に見てもらうというのは、なかなかいいバランスだと思っています。

## 22.5　サービスが手のひらから飛び立つとき

ここまで読んでいただいてありがとうございます。最後に、モバイラーズオアシスを運営していて経験した素晴らしい思い出を書いてまとめにしたいと思います。

モバイラーズオアシスに寄せられる情報は、スパム防止のため管理者の承認が必要となります。そのため、たいていの情報は頭の中に入っていて、モバイラーズオアシスを開かなくても「この駅で電源が使えるのはこのお店」と把握できてしまいます。「自分のために作ったサービスなのに、意外と自分では使わないんだな」と複雑な思いを抱きました。

ところが何年か続けると「これはまだ載っていないはず」と登録しようとしたお店が、すでに掲載されていて驚くという事件が起きました。年が経つにつれてその割合は増えて「これも載ってたのか」「これは載ってないと思ったのになあ」「こんなところのお店まで載ってるの！？」と思うことが増えたのです。この現象はずっと続き、ついには「このサイトすごくない？いったいどこからこれだけの情報が集まってくるの！？」と思うようになりました。自分のサイトなのに！

この本は個人開発者の本ということで、1人でサービスを開発する人たちが集まって書いた本です。個人開発したサイトは、最初のうちは自分しか使う人がいなくて、つまらないなあ、と思ってしまうこともあるかと思います。けれども、ユーザーが増えてくると、次第に、予想もしていなかったような場面で使われたり、思いもしなかったような使いみちを思いつく人が出てきたりします。

自分の手から生み出されたサービスなのに、いつの間にか開発者の手から飛び出して、サービスがひとりでに歩き出していくのです。これは他の趣味ではなかなか味わえない感動です。ぜひあなたも、サービスを開発してこの感動を味わってみてください。

```
mogya / @mogya

サラリーマン時代に友人たちと開発したサービス「いいめも」のヒットをきっかけに独立。
フリーランスとしてプログラマをしつつ、街の電源検索サイト「モバイラーズオアシス」
を運営。フロントエンドやサーバサイドの開発はもちろん、登記や法人決算、台所水栓の
取替まで自分でやっちゃうフルスタックエンジニア。

https://oasis.mogya.com
```

1. https://oasis.mogya.com/

# 第23章　Build to Think！

　ウェブエンジニアの大日田 @ohida と申します。2005年から個人サービスを開発・運営しています。振り返ってみると僕は「Build to Think」という考え方を大切にしてきたように思います。個人開発をはじめようとする方に向けて「Build to Think」を紹介します。

## 23.1　15年間を振り返って

図23.1: 15年間を振り返って


1．もともと絵を描いたりものをつくったりすることがすごく好きでした。

1．インターネットによって世界レベルで情報のやり取りができることに感動しました。

1．自分が書いたコードが画面上で動くことに興奮しました。

1．個人ウェブサービスづくりを通して自分の力で多くの人に何かを届けられる喜びを知りました。

図23.2: コトノハ - TOP画面


図23.3: コトノハ - 利用画面


「コトノハ」[1]というウェブサービスは、一週間くらいで開発したものをリリースしました。そのため、UIをはじめとして、多くの点に課題がありました。利用者の方からコメントをいただき、日々改善を行ったことを覚えています。とても楽しく貴重な体験でした。対話的な開発は利用者と開発者双方の熱量を高めるのだと実感しました。

## 23.2　つくりたいからつくる

　個人開発をするのには様々な理由があると思います。根底にあるのは「何かつくりたい」という衝動ではないでしょうか。

　僕の場合は、新しいウェブ系技術への興味関心がきっかけとなることが多いです。テクノロジーの誕生でこれまでできなかったことができるようになる。その意味において、テクノロジーの誕生は、新しい可能性の誕生とイコールです。

　常に新しい技術がうまれているウェブの世界は、常に新しい可能性に満ちていると言えます。そんな無限とも言える可能性の中にいるのです。自分に何ができるか？自分は何をしたいか？自分ならすごいことができるのでは？そんなことを考えるのはとてもワクワクします。

　まるで目の前に無限に尽きないおもちゃ箱があるようなものです。おもちゃ箱をひっくり返して、積み木やブロックやクレヨンなどを前にして「さぁ遊ぼう！」と思っていた子どもの頃の感覚と、個人開発のワクワク感は、とても近いと思っています。

## 23.3　個人開発と創意工夫

子どもの遊びというのは創意工夫に満ちています。おもちゃや遊びには、正解や不正解というものがありません。

粘土でなにかつくるとき、事前に綿密な完成予想図や設計図を用意し、正確にそれを再現する、ということはしません。まずは手で粘土をこねくりまわしながら、あーでもないこーでもないという感じで、それでも何か自分らしいものをつくりあげていきます。

決まった正解のある世界では、創意工夫の余地はありません。正解がない世界では、それでも先に進むための創意工夫が求められます。僕が思う「個人開発者」は、つくることに興味がある人のことです。正解がない状況における創意工夫が好きであり、得意なのではないでしょうか。

## 23.4　Build to Think

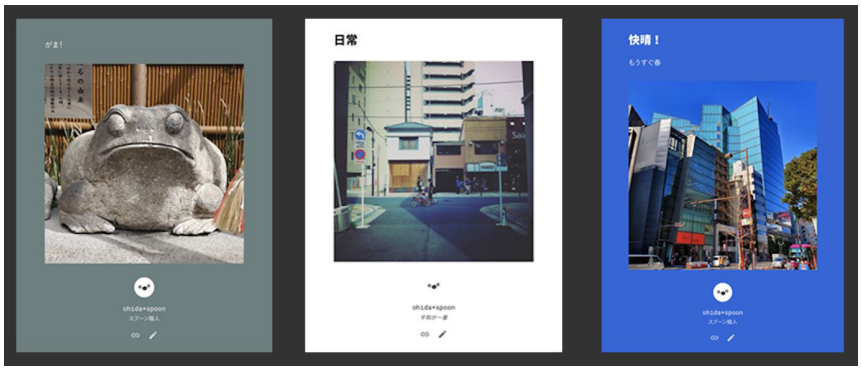
僕が好きな言葉に「Build to Think」というものがあります。これはつくってから考えるプロセスのことを言っています。たまに勘違いされるのですが、考えることをあとまわしにしてつくることを正当化する、という話ではありません。文字通り「考えるためにつくる」ということです。

かたちあるものをつくってみるというのは、人間の考える力を最大限に引き出すための有効な手段です。人というのは頭の中だけで完全な思考ができるわけではありません。子どもが粘土で遊ぶときには、ひたすらに手を動かしながら、あれこれと試行錯誤して理想のかたちに近づけていきます。プロダクトづくりも同じだと僕は思います。

実際にものをつくってみると、思ってもみなかったような気づきやアイデアが生まれてくるものです。企画とロジックとUIをどうやって最適なかたちにするか。まずつくってみて、それをもとに考え、またつくる。そんな自己対話的なものづくりができるのが個人開発の面白さではないでしょうか。

## 23.5　オフラインで閲覧できるブログ

図23.4: PWA x ビジュアル x ブログ




2019年は「オフラインで閲覧できるブログ」を作っています。オンラインでサイトを閲覧するという体験が今のウェブでは主流です。一味違う体験ができたら面白いのではないでしょうか。

もともとはPWA（Progressive Web Apps）という技術でウェブアプリでもつくってみようと思いました。あっという間にオフラインでも閲覧できるブログが実現しました。

せっかくのSPA（Single Page Application）なので、画面遷移のエフェクトをかっこよくしたくなります。するとエフェクトをスムーズに動かすためにサーバの応答処理速度を改善する必要が出てきました。

さらに次は、SNSでシェアしたときにビジュアルを訴求できるように、OGPに対応したくなります。レンダリングの処理を工夫することで実現しました。

そして同じように気付きと開発のサイクルが周り続けます。これが Build to Think です。ものづくりを通して、さらなるものづくりのヒントに気付くことができます。

## 23.6　Build to Think のサイクルを広げる

Build to Think の考え方でものづくりを行うと、さらに素敵な展開があります。

Build to Think的に言えば「完成」というのはもうそこにThinkの余地がない、という状態を指します。つくって考えてまたつくるのが好きなクリエイターにとって「完成する」ということはなかなかありません。もしそうなったときにはそのおもちゃに飽きてしまうのではないでしょうか。

なのでプロダクトは「完成に至らない状態で出すべき」だと僕は考えています。ウェブサービスにしてもアプリにしても、つくったものを世の中に出すのはとてもドキドキします。つくる中でやりたいことが増えて、リリースがのびのびになってしまうことも、よくあります。

それでも、やるべきことは、早く誰かに使ってもらうことです。そして利用者から意見をもらうことです。これはBuild to Thinkのサイクルを個人から外の世界に広げるということを意味しています。

## 23.7　みんなでThink！

開発段階でのBuild to Thinkで活性化されるのは開発者個人の考える力です。しかし、サービスをリリースすれば、利用者みんなが目の前にある「それ」のことを考えることができます。

つまり、サービスを利用してくれる人たちみんなをBuild to Thinkのプロセスに巻き込めるということです。そこには個人の思考の限界を超えた大きな可能性があります。これはまさにインターネット的なものづくりのかたちだと思います。

正解やゴールがないものについてみんなで考える、というのは素晴らしいことではないでしょうか。個人の言語化できないような衝動から生まれたものが、試行錯誤を通してひとつのかたちに結実し、そしてその利用者たちも巻き込んで際限なくかたちを変え進化していく。

そういう可能性を持っているのが、個人開発の大きな魅力です。

つくりたいからつくる。僕のような個人開発者にとって「Build to Think」とは、前に進むための勇気と力を与えてくれるキーワードです。同時に、自分自身や利用者との対話的プロセスを核としたものづくりのパラダイムなのです。

## 23.8　おわりに

　これまでつくってきたサービスのことや自分の開発のあり方を振り返ってみると、自分にとってはBuild to Thinkがとても重要な概念だということにあらためて気づきました。

　画家のパブロ・ピカソも「何を描きたいかは、描きはじめてみなければわからない」と言っています。何をつくっていいかわからないときこそ、とりあえず何かをつくってみてはいかがでしょうか。

　個人開発というのは最高に面白い世界ですので、ぜひ多くの人に挑戦して欲しいと思います。この記事が少しでもその後押しになれば嬉しいです。[2]

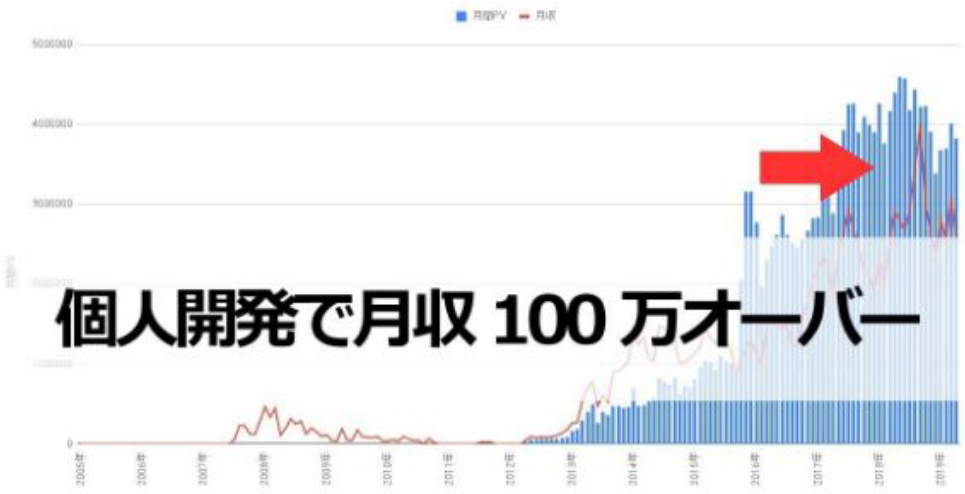
```
オオヒダタカシ / @ohida

こんにちは。スタートアップから大手企業において、新規事業開発やウェブサービスの企画・開発・デザインを行なっています。
世界を良い方向に変えていく力を持ったプロダクトをつくりたいと思っています。

http://solvalou.net/
```

1. http://kotonoha.cc/
2. 本稿は執筆者（@ohida）の書き下ろし文をベースに、ブログ記事『プロダクトマネージャーについて語るときに僕の語ること』の内容を一部抜粋して編集したものです。

# 第24章　14年かかった！個人開発で月収100万達成した話

　こんにちは！専業で個人開発しているSiRO（@codemesi）です。苦節14年、個人開発で月収100万円を達成しました。便利ツール系のサービスを中心に提供しています。これまでの知見を共有するので、同じ道を志す方の参考になれば幸いです。

図24.1: ヘッダー画面

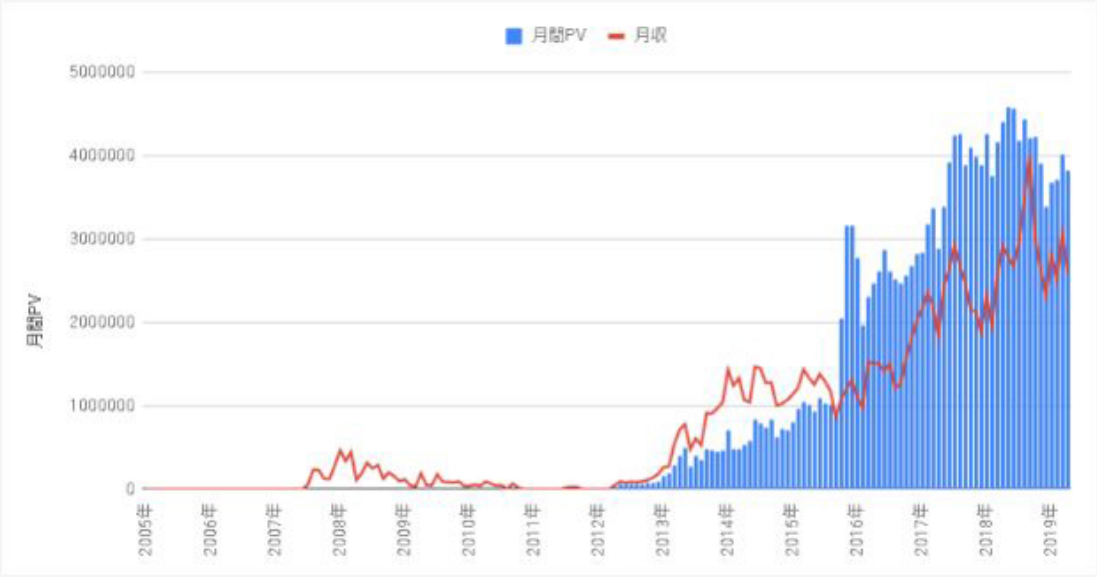


## 24.1　自己紹介

　SIerを10年くらいやって、今は会社を辞め専業で個人開発しています。労働に追われる人生が嫌で、とはいえ起業する野心もないし、人の上に立つ人間でもない……。不労所得となる資産を作って「毎日自由に、のんびり生きていく」のが目標で、個人で色んなWEBサービスを作ってます。

## 24.2　実際の月間PVと月収

図24.2: PV画面




これは全サービスをサマリーした数字です。収入は全て広告によるものです。

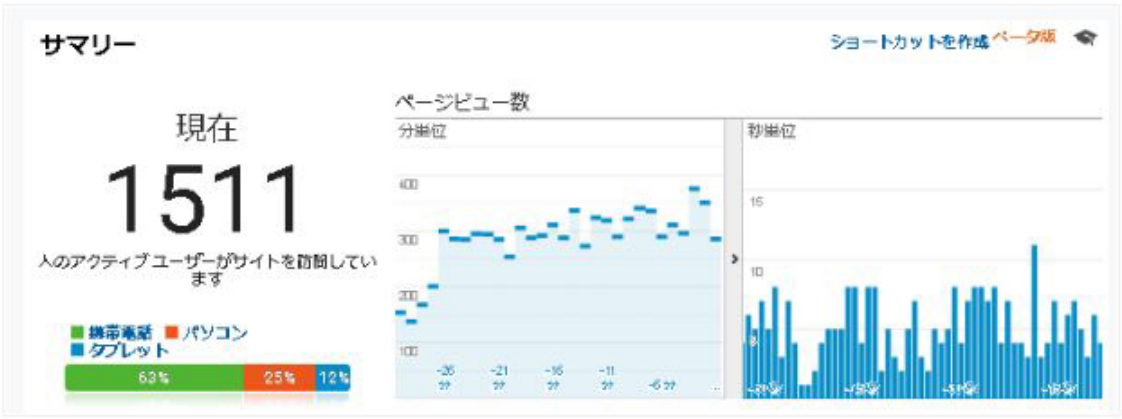
PVは最高記録で450万PV/月くらい。月収の軸ラベルは隠してありますが、ピーク時に100万円オーバーです。月収とPVの関係をまとめると以下のようになります。

| 月収 | PV |
|---|---|
| 月収10万円達成 | 29万PV |
| 月収20万円達成 | 48万PV |
| 月収30万円達成 | 70万PV |
| 月収40万円達成 | 267万PV |
| 月収50万円達成 | 282万PV |
| 月収60万円達成 | 339万PV |
| 月収70万円達成 | 440万PV |
| 月収80万円達成 | 444万PV |
| 月収100万円達成 | 422万PV |

PV数は参考程度の数値です。特にツール系は何をもって1pvとするかは作りによって大きく変わります。

## 24.3　バズった経験

図24.3: サマリー画面




小規模ながら、何回かSNSでバズってこともあります。バズといっても色々なケースがありました。

図24.4: インフルエンサーが紹介してくれてバズったケース

| | PV | 訪問数 |
|---|---|---|
| 前日 | 713 | 48 |
| バズ当日 | 90,771 | 8,252 |
| 1日後 | 96,346 | 4,009 |
| 2日後 | 90,053 | 2,851 |
| 3日後 | 79,398 | 2,520 |
| 4日後 | 76,701 | 2,192 |

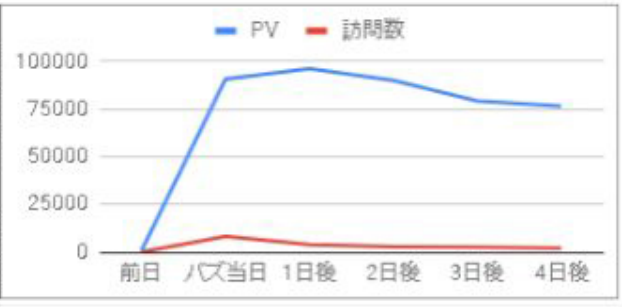



図24.5: リリース直後にバズったケース

| | PV | 訪問数 |
|---|---|---|
| 前日 | 0 | 0 |
| バズ当日 | 12,065 | 8,944 |
| 1日後 | 19,403 | 13,821 |
| 2日後 | 4,403 | 3,409 |
| 3日後 | 1,794 | 1,402 |
| 4日後 | 483 | 378 |

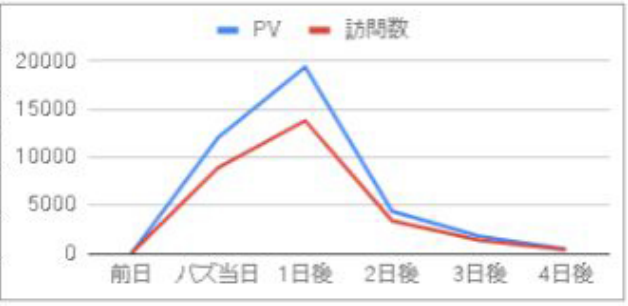
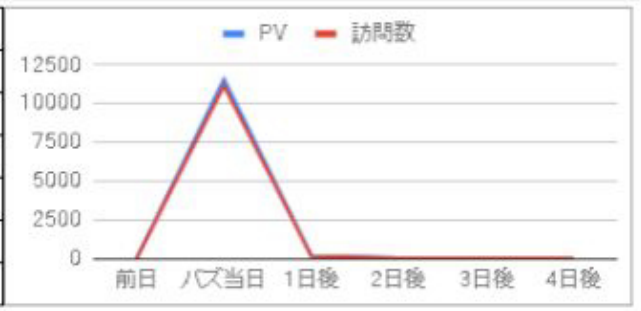



図24.6: 一発屋で終わったケース

| | PV | 訪問数 |
|---|---|---|
| 前日 | 0 | 0 |
| バズ当日 | 11,567 | 11,164 |
| 1日後 | 177 | 169 |
| 2日後 | 14 | 14 |
| 3日後 | 6 | 6 |
| 4日後 | 2 | 2 |



## 24.4　ランニングコスト

　全部でおよそ月に2〜3万円くらいのランニングコストです。AWS、GCP、VPS、レンタルサーバが15台くらい。ドメインが10個くらい。すべて一人でやっているので外注費はかかっていません。

## 24.5　月収100万円で見えてきた事

月100万いったら遊んで暮らせるのではないか、と最初は思っていました。しかし実際に達成してみると、少し事情が違いました。

まず、システムメンテナンスや保守、ユーザーサポートなどの作業が発生します。また、そもそも収入が将来も安定する保証はどこにもありません。リスク分散するため「マネタイズを分散しよう」とか「別サービスを作ろう」という発想になっていきました。結局どこまでいっても安泰はないです。

ただ、会社員時代に比べ、生活にゆとりが出来たのは間違いないです。起きる時間も自由だし出勤もないし、仕事をするしないも全部自由です。

少し周囲の反応も変わってきたように思います。法人から集客の相談をされたり、某大企業からサービスについて話を聞かせて欲しいと言われ、出向いたこともありました。

## 24.6　失敗を重ねて得た知見

　月100万円に到達するまでに結局14年かかりました。特に最初の5年は失敗続きでした。作っては壊しを繰り返したり、儲かっても続かなかったり。しかし失敗を重ねるうち、徐々に照準があっていき、継続して稼げるようになっていきました。体験談や知見を書いていきます。

## 24.7　アイデア編

ユーザー投稿型サービスは難しかった

　ユーザー投稿型とは掲示板やSNSのようなサービスです。ユーザーがたくさん集まらないと面白くならないような投稿系サービスは、ほぼ失敗しました。放置してもユーザーは増えないし、無名で予算も無い個人だと、集客能力が無くて難しいと感じました。

　それからはいきなり投稿系サービスを作らず、やるとしてもサブ機能として投稿機能を持たせるようにしています。

　投稿系サービスを諦めて、代わりに便利ツールを作ることにしました。これならユーザーがいなくても価値を提供できると考えたからです。

ツール型サービスは手堅い

　検索エンジンからの流入もあり、わずかながらユーザが増えていきました。ユーザー投稿型と違い、放置しても、ツールとして価値のある状態です。これはアリだなと気づきました。

　放っておくとどんどんPVが増えました。モチベーションも上がり、機能追加も積極的にしていくと、さらにPVが増え、良いサイクルになっていきました。

　ここで投稿機能をつけました。一度は諦めた投稿機能ですが、今度はユーザが投稿してくれるようになりました。いきなり投稿系サービスを始めるのではなく、まずツールとして開発し、ユーザーを集めてから投稿を促す。その方が投稿されやすいことを学びました。

みんなが欲しがるものを探す

　SNS上でみんなのツイートを普段からダラダラ見るようにしています。「何に興味があるのか」「何に不満があるのか」「何を面白がっているのか」を観察しています。じゃあこういうの作ればウケるかも！という発想を出していきます。

　中にはド直球で「こういうのあったら便利なのに」というツイートがあります。RTが伸びているのを見て「なるほど～そういう需要があるのか」と思いサービス化したこともありました。需要があるのは分かっていますので、リリースするとやはり反応は良かったです。「作ってみました！」とその人たちに返信すれば、ほぼ間違いなく広めてくれます。その界隈の人たちが喜んで最初のユーザーになってくれます。

　もっと極端に言うと、サービスをインフルエンサーに最適化する方法もあります。その界隈で影響力ある人のツイートなどをチェックし、彼らの好みそうな機能やコンテンツを作るのです。いつしか彼らの目に留まると高確

率で拡散してくれます。このように「自分が作りたいものを作る」のではなく「インフルエンサーが欲しがるものを作る」といった発想をすることもありました。

他にも、利用者目線でかゆいところに手が届くサービスを考えたり、この技術やデータを使うとこういうサービスを作れるかもと発想したり、競合サービスを参考にして自己流のアレンジを加えたりしました。

### 自分が続けられるジャンルを狙う

自分が得意なジャンルで闘うのがベストだと思います。それほど思い入れのない分野で、いかにも世の中の需要を分析した気になって、儲かるだろうと狙って作ったものは全部失敗しました。

詳しくないジャンルを選べば、業界のリサーチがゼロから必要になります。失敗して当たり前でした。それに、お金目的だけだと、すぐに利益が上がらなければモチベーションを保てません。すぐに辞めてしまいます。

あとは、少しニッチな穴場を狙うようにしています。ジャンルを大きくして大手サービスと勝負するよりも、参入者の少ない領域で勝負しています。

### 1つのジャンルを掘り下げる

私はスーパーマンではありません。ジャンルの異なる多種多様なWEBサービスを、思いつきで作っていき全部ヒットさせることはできません。それは金なし知名度なしの個人では不可能だと思います。

これまでと全く畑違いのサービスを作るとゼロからのスタートになります。しかし、似たようなジャンルであれば、それまでの資産を活用できるのです。資産というのは登録ユーザのリストやデータ、ノウハウ、ソースコードなどです。

このDBを流用すれば少しの機能追加だけで新しいサービスが出来るのではないか。しかも既存ユーザに宣伝もできる。こういった資産を、横展開する発想です。

これを続けるうちに、そのジャンルに精通していきました。新規参入者であれば数カ月かかるであろうサービスを、自分なら数日で作れてしまう。そのような優位性が生まれました。ジャンルをあまり広げず、過去の資産をどんどん活用していくことで、効率的にサービスを増やしました。

## 24.8　システム編

サービス設計

以下のような検討をしてから実装に入ることが多かったです。

・自分の強みは？（参入ジャンルの選定）
・アイデア出し
・ターゲット層は？（利用者のイメージ）
・サービスのウリは？強みは？
・競合は？競合との差別化は？
・マネタイズ方法
・各機能の検討
・サービスの利用イメージ、ユースケース
・集客アイデア
・SEO、SNS対策
・技術選定、アーキテクチャ検討
・初回リリースの範囲を決める
・画面イメージ
・画面の洗い出しと簡単な設計
・バッチの洗い出しと簡単な設計
・DB設計

綺麗なドキュメントは不要なので、メモ書き程度の設計をしています。資料のメンテはしていきません。

実装・コーディング

SNSやブログなどでアウトプットは一切せず、プロダクト開発のみに集中するスタイルでした。内向的な性格もあったり、あれこれと並列作業をするのが苦手なタイプのため、会社員時代は帰宅したらずっとコーディングしていました。

最初のころは技術力が高ければ稼げるのだろうと勘違いしていました。イケイケの技術や美しいコード、大規模サービスの知見とか、仕様変更に柔軟なクラス設計などなど。それらを求めても自己満足で終わってしまい、あまり意味がなかったです。なるべくシンプルな作りにして、スピード重視という開発スタイルに自然と変化していきました。

企業での大規模開発ではメンバーが増えることで多大なリソースがかかります。共通処理をどうするとか、作りがバラバラにならないようにとか、IF部分の意識合わせとか、仕様決定待ちとか、情報共有とか、あの人はもう離任したとか。そういった雑多な作業がないので、個人開発のほうが生産性の高さを実感します。

## 24.9　保守はツライ

保守も全部一人でやるわけですが、規模が大きくなってくるとツライです。保守の発生要因は、主に老朽化と外部連携でした。長くシステムを運営する以上、老朽化への対応はある程度仕方ないと思っています。

外部連携というのは、例えば呼び出しているAPIの仕様が変わった、といったケースです。各種クラウドやサーバ、API、スクレイピングなど外部システムとの連携です。仕様が変更されると、自分のシステムも影響を受けます。これらは全て保守作業の量に直結しました。

そこで、本当に自分に必要か吟味して、慎重に技術を採用するようになりました。世の中に出回る技術情報はほとんどチーム開発が前提だったり、保守やコストの観点が抜け落ちています。個人開発にとってはオーバースペックです。

AWSなどのクラウドも魅力的に見えますが、個人で運営するには料金が高くなりがちです。私は「安いVPSで十分ではないか」と判断することが多いです。

## 24.10　メンテナンス編

サービスのメンテを減らす

　個人運営なので、他の仕事やプライベートが多忙になったりすると、運営まで手が回りません。副業で個人開発をする場合はなおさらです。

　メンテしきれずにそのままサービス閉鎖につながるケースは多かったです。多忙な生活の中で、しかも儲かっていないシステム。これをメンテし続けていくというのは相当の精神力が必要です。自分にはとても無理でした。

　そこで、保守作業が多くなりそうなサービスは作らない方針に転換しました。多少クオリティを犠牲にしても閉鎖することはほぼ無くなりました。サービスの数を増やしていくことも可能になりました。

ユーザーからの意見をもとに磨き込む

　ツイッターのフォロワー数や、サービスの登録数は合計で数万の規模に成長しました。ユーザーからの意見は参考になります。いただいた指摘に即座に対応すると、そのスピード感に驚かれるし、すごく喜ばれました。

　中には熱心に指摘してくれる人もいました。他にも何かありますか？と聞き続けると、次々と新しい要望を出していただきました。「この人スゴイ」と思い、何カ月か費やして、この人の要望にすべて対応しました。たった一人のユーザーの意見ですが、この時期にサービスが一気に進化したと思います。

## 24.11　マーケティング編

### リリースは通過儀礼

最初のころは公開して一発当てるという発想でしたが、上手くいきませんでした。リリースはスタートラインです。サービスを育てるという発想に変えました。

ブログやSNSやSEO、様々な外部サービスで宣伝を続けて、じわじわと利用者を増やしていきました。無料のプレスリリースを各社に送信しまくったら、採用されて記事にしてくれたこともありました。

### リピーター

SNSでバズったり、SEOで順位が上がって、ユーザーが増えるのは刺激的です。しかし、集客効果は一時的なもので終わってしまい、長続きしないケースが多かったです。

長期的に安定して稼ぐには、直接サイトに来てくれるリピーターが重要だと思うようになりました。

・定期的にコンテンツを更新し、また見に来る理由を作る
・ブックマークを促す
・SNSのフォローを促す
・RSSの登録を促す
・WEBプッシュ
・アプリのインストールを促す
・ユーザー登録を促す
・メールアドレスの登録を促す
・メルマガ送信で自サービスに誘導する

コアなファンを地道に作り、サイトにまた来てもらうために、このような施策を打ちました。

### 人気サービスになると……？

いつの間にかニュースサイトでサービスが紹介されたこともありました。たまたま本屋で立ち読みしていたら自分のサービスが載っていてびっくりしたこともあります。事前に掲載連絡はなかったので、自分で調べて初めて気付きました。

## 24.12　トラブル編

削除依頼

　たまに他人の誹謗中傷やプライバシーなどをサイトに書き込まれることがあります。関係者やその弁護士から「削除してください」という依頼が来ます。大きなトラブルに発展したことは無いですが、迅速に対応するように気を付けています。

スクレイピング

　安易にスクレイピングをしてしまい、一部データを転載したところ某社から怒られてしまいました。そのデータをスクレイピングしている人達は他にもたくさんいたし、多分大丈夫だろうと考えたのが甘かったです。

　対応しないと法的措置とのことでしたので、迅速に対応し謝罪したところ、お許しをいただけました。迅速に対応したことは評価され、むしろお礼を言われました。

　ややこしいのですが、スクレイピングしていたサイトから怒られた訳ではありません。そのサイトにデータを提供している大元のデータ所有者の企業から怒られたのです。

　だから最初は「ん？こんなサイト、スクレイピングしていないぞ！？」と困惑しました。それ以来、各サービスをまたがるデータの流れや情報提供元を意識するようになりました。

JavaScript製ツール

　マネタイズ手段としてCoinhive[1]の導入を検討したのですが、そのあと事件が話題になりました。最近はJavaScriptの利用方法を誤ると逮捕リスクが出てきているようなので注意しています。

## 24.13　マネタイズ編

広告によるマネタイズ

　Googleアドセンスの収益性は圧倒的に強いです。でも規約が厳しいので、指摘を受けたら迅速に対応していく必要があります。

　ASPなどのアフィリエイト広告は、自分には合わなかったです。サービスにマッチする広告がなかなか見つかりませんでした。無理にマッチしない広告を貼っても、ほとんど儲かりませんでした。

　自由度が高い上に、継続契約によって収益が安定するのが、純広告です。競合サービスや近いサービスがどんな会社の広告を貼っているか参考にしています。

　いきなり堂々と広告枠を販売しても、なかなか返事をもらえません。まずは試供品として広告枠を格安で提供します。相手を先に儲けさせてあげることで信頼関係を作り、そこから正式に販売すると成約することができました。最初は必死に売り込まないといけませんが、一度関係を作ってしまえば、向こうから連絡がくるようになりました。

「まだマネタイズしない」という選択肢

　サービスを企画する時「これ作ったら絶対面白いんだけど、マネタイズが難しい」という局面もありました。

　イメージ通り実現できるかも分からないし、本当にウケるかも分からない。どのくらいの規模でユーザが集まるかも分かりません。そのような場合は、まず作り始めました。

　作ってみたら予想通り反応が良く、すぐ登録ユーザが1万人以上になったサービスがありました。一時的に広告主がついたこともありました。残念ながら現在は利益ゼロで運営中です。

　無計画な進め方ですが、ひとまず反応がいいモノを作れたのでヨシとします。利用者がもっと増えたら課金などを真剣に考えていこうと思っています。

先に価値を提供する

　「価値を先に提供し、利益の回収を少し遅らせる」と上手くいく事が多かったです。

　ツイッターでいきなり自サービスを宣伝するのではなく、アカウントを育てて認知度を高めてから自サービスを宣伝する。

　メルマガを出すときはすぐにアフィリエイトを貼るのではなく、有益な情報をたくさん与えて信頼を得てから広告を貼る。

　サービスをリリースしてもすぐに広告を貼らずに、サイト内をよく循環してもらい、しっかりとリピーターになってもらってから広告を貼る。

この寄稿のマネタイズ？

　この文章も無償で提供しています。noteに公開した記事を元に編集いただいています。編集された文章に問題がないかレビューしたくらいです。

　特に目先のメリットがあるわけではありません。価値を提供すれば、いずれお金や何らかのメリットに変わると思っています。

　どのような利益になるのかは分かりません。何がウケるのかも分かりません。とりあえず色々な価値を世の中に提供してみる。その中で一番反応が良かったものを伸ばしていけばいい。

　なので、この寄稿もチャレンジの一環です。少しでも参考になる所があったら幸いです。今後も個人開発を頑張っていきたいと思います。[2]

```
SiRO@個人開発 / @codemesi


個人開発で食ってます。月間400万PV、月収100万を達成。

SIerを辞めていろんなWEBサービスを作っています。

エンジニア歴17年、個人開発14年目。


https://note.com/codemesi
```

1. サイト閲覧者に仮想通貨をマイニングしてもらうツール。CoinHive事件として賛否両論を巻き起こす裁判に繋がった。サイト閲覧に応じて収益が上がる仕組みを広告以外で実現できるため、日本中のクリエイターたちから期待を寄せられていた。
2. 本稿は執筆者（@codemesi）による同名のブログ記事を加筆・修正して寄稿したものとなります。

# 第25章　それでも挑戦は止められない。個人開発の絶望と希望

## 25.1　純粋な個人開発は楽しい

@dala00 と申します。プログラミングが好きで、色々なサービスを作ってきました。

個人で何かを作るというのは非常に楽しいものです。私は高校の頃からプログラムをはじめ、毎日PCに向かってプログラムを組んで遊んでいました。

試行錯誤しながらなんとかプログラムを組み、動かす。PCには元々すごいソフトがたくさん入っているのですが、自分ではじめて書いた数行のプログラムが、それらと同じ環境で動くというのは本当に感動ものでした。徐々に上達していき、更に色々な事ができるようになると、もっと楽しくなりました。

大学4年になると更に上達していました。まだ2003年頃で、私はその年インターネットをはじめたばかりでした。ようやくプログラミングの知識を本以外で得られるようになったところだったので、今の若い方達よりはできないことが多かったと思います。

とはいえ個人でゲームを作ったり、ウェブサイトを運営したりして楽しみながら色々な事を学んでいきました。思えばこの頃が一番何のしがらみもなく、ただ純粋に個人開発の楽しさを味わえていた時期だったのかもしれません。

## 25.2　目的に向かって進むための個人開発

そんなこんなで時は過ぎ、私も社会人となり働いていました。就職活動はしていなかったので、警備員のアルバイトをしながらプログラミングの仕事を探し、なんとか見つけた会社でプログラマーとして働きました。社会保険にも入っていませんでしたし、完全新人ということで給料も非常に安かったです。とはいえ、今考えると新人でちゃんとプログラミングを学べるというのは非常にありがたい環境だったような気もします。

そんな中で恋人ができ、付き合って1年ですぐ結婚しました。幸せなことですが、当時の自分の仕事は収入も少なく、わりと厳しい生活を送っていました。

なんとかしなければと思い、何かネット上でお小遣い稼ぎみたいなことはできないか調べました。すぐに大手のお小遣い稼ぎ紹介サイトが見つかり、そこでアフィリエイトを知り、自分もお小遣い稼ぎ紹介サイトを作ってアフィリエイトをはじめました。

でもこういったものは基本的にうまくいかず、一部の大手のサイトだけが稼げるような形だったため、だらだらと続けつつそのうちお小遣いサイトは止めてしまいました。

代わりに、APIを使ってAmazonアソシエイトを利用した無限に本を紹介するサイトを作ったりして、その頃からプログラムを使ってお小遣いを増やそうという試みを続けていました。ただ、やはり基本的にはうまくいかず、収入は全く増えませんでした。

でもやはり少しでもお金を稼ぎたい、という思いは続いていました。お金があれば家族で安心して笑顔で暮らせるからです。この頃は会社をやめてフリーのWebエンジニアになっていました。ただ、やはり給料があるわけではないので先行きは不安でした。先の見えない不安に満ちた厳しい生活から速く抜け出したいと思っていました。

## 25.3　Webサービス作りをスタート

Webサービス期

　他のアフィリエイターと比べ、プログラムを使えるところが強みなので、広告をうまく設定して稼げるようにWebサービスをどんどん作りました。当時はレンタルCGIサービスが流行ったので、私もためしに運営していました。集客のためにレンタルカウンター、広告設置のため無料ホームページスペース、レンタルブログ等を開発・運営しました。

　結局人は集まらず、全くうまく行きませんでした。しかも、普段使いのPCをサーバーとして使っていたため、セキュリティもきちんとしておらず、悪意を持ったユーザーにPCに侵入され、ほとんどのデータを削除されて終わることになりました。そのちょっと前にAdSenseも大量連打されてBANさせられていました。その後もファッション系SNS、デートプラン作成ツール等を作りましたが、うまく行きませんでした。

ソーシャルアプリ期

　そんな中、mixiがソーシャルアプリを開始しました。まだ開拓されていない分野で、誰でもアプリを作れるため、みんなが熱狂して色々なものを作っていました。法人向けのプランもはじまり、色々な会社が参入しました。日本ではまだ新しい領域でユーザーの食いつきも良く、課金率も高かったように感じました。それなりのものを作れば成功できると思い、私も法人化して参入しました。mobageやGREEもソーシャルアプリを開始しだしたので、うまく行けば横展開してかなりの収益を得られると思いました。

　小説アプリ、MUSICアプリ等を作りましたが、メインとなるのはネココレースというソーシャルゲームでした。こつこつプレイして成長させていくゲームで、ネココさんというキャラクターに着せるアバターを有料アイテムとして販売しました。でもやはりうまく行きませんでした。ほかのアプリも直接広告主を募集して収益を得ようと思いましたが、結局広告主は見つかりませんでした。

スマホアプリ期

　その後、テラバトルというゲームがリリースされ、爆発的な人気になりました。全体的にクオリティは高いのですが、ゲーム自体は凄くシンプルでした。これなら私が作ったものもうまく売れるのではないかと思い、試しに似たようなものを作ってみました。

　さすがにここまで多くの失敗を経験すると、自分自身集客が下手だというのはもうこの時点で気づいていました。そのため、集客に困らないようYahoo! モバゲーとmobageのスマホバージョンでハイブリッドリリースしました。

しかしやはりだめでした。カンストまで無茶苦茶はまってプレイしてくれるユーザー等も現れ、嬉しかったことは嬉しかったのですが、収益はほとんどありませんでした。mobageのアクセス数が分からず怖かったこともあり、月1万円のクラウドサーバーを利用したため、赤字でした。せっかくの10連ガチャは一度も売れませんでした。

諦めきれず他にもネイティブで直接ライトなRPG系のゲームや子ども向けアプリをリリースしてみました。それらも全てうまくいきませんでした。

個人開発をはじめて7〜8年くらいでしょうか。他の事にハマって何も作ってない時期もあったので、この間ずっと開発し続けていたわけではありませんが。

個人開発を始めた時は、いつか大ヒットするサービスを作って、お金持ちになって、家族みんなで楽しく過ごす、そんな楽しい未来を夢見ていました。

でも実際にはその様な未来は訪れませんでした。ひたすら失敗ばかりです。何を作っても、どんなジャンルのものをどんなプラットホームで作っても、全く人は集まりませんし、売上もほとんどありません。それが当たり前になっていました。私の作るものは全てそういうふうになるしかないのだ、という現実が当たり前のことになっていきました。

僕はもう一生成功することはないかもしれないな、という絶望と共に生きていました。

## 25.4　転機

とはいえ、趣味でもあるので、なんだかんだでサービスは作り続けていました。ふと急にアイデアが思い浮かび「もしかしたらいけるんじゃない！？」という気持ちも時々湧き上がりました。2017年の年末は「ヒットするまで色々作り続けるべきなのでは！？」と思いたち、Elixir&Phoenixを使って「ひたすらサービスを連続で作り続けよう」というモードに入って、とにかく立て続けに開発しました。

そんな中、実際にお金を稼ぐための手法はどこかに転がっていないか、という事を考えていました。且つ、そのころSlackが流行りはじめていたため、もしかするとマネタイズをうまくやれる人が集まったSlackグループがあるのではないかと思い探してみました。

すると「個人開発者マネタイズの会」というグループを発見しました。[1]ずっとソロ活動で人と関わるのは得意ではなかったのですが、もうそんな事を考えている場合ではないということはよく分かっていたため、ワラにもすがる思いで参加しました。

そこはどちらかというとみんなで頑張ろう、というグループだったため、一気に何かがうまくいく、ということはありませんでした。でも、一人で何かをやっているよりはみんなで切磋琢磨がんばれる方が安心感がありました。サービスをリリースした時もアドバイスを貰ったり、逆にアドバイスしたりすることでお互いに学びがありました。

その後のある日、Twitter上で「運営者ギルド」なるSlackグループが発足したことが話題にあがりました。[2]サービス運営者が集まるグループ。とにかくWebサービス運営についての情報がたくさん欲しかったため、すぐに参加しました。

Slack運営のスダさんはすでに若くして自社のサービスで生計をたてており、参加者の中にも自社サービスで頑張っている人がおり、有益な知見が飛び交う場所でした。そこにいるだけで有用な情報と学びを得ることができました。

## 25.5　一番大きな学び

一番有益で、大きかった学びがありました。それは「努力すること」です。

私は別に努力を知らなかったわけではありません。むしろ技術のインプットが好きなWebエンジニアですし、努力してスキルアップして前に進むことは大好きです。普段から人一倍努力はしていました。でも、私の中ではWebサービスの運営に関してはどれだけ努力をしても何も実らないものになっていました。

自分の目から見えるWebサービスの成功者というのは、とてもキラキラしていました。みんな若くて、初めて作ったサービスでいきなり大成功していて、カリスマ性もあり、誰からも称賛される、生まれながらの勝者みたいなイメージです。

だからそれと比べると、私は一人の人間としてはキラキラした魅力もカリスマ性もあるわけではなく、むしろ根暗で近寄りがたく、どんなサービスもうまく広めるだけの影響力もない、あきらかにその勝者達とは程遠い存在でした。

でも、色々とSlackに流れる話を聞いていると、スダさん達のサービスは成功させるまでに1年以上かかったそうです。その間自分たちでコンテンツを作ったり、サービスを改善したり、努力を続けていたということでした。さらに、実はその前から多くのサービスを作り、失敗を繰り返していたようです。ぱっと見はキラキラした成功者というイメージがありましたが、全くそうではありませんでした。成功の裏には努力があったのです。

それからというもの、私が日々得る情報のなかで、そういった努力の元に成り立つ成功例がよく目に入るようになりました。世の中に溢れているものは、キラキラした特別な存在ばかりではなかったのです。裏にある苦労に目を向けず、勝手に表面上のキラキラばかりを見て絶望していただけだったのです。

安易に簡単にお金を稼ぐ方法があるのではないかと考えて、努力を放棄していました。それが全ての敗因だったという事が今はわかるようになりました。

もちろんサービスを作ること自体が努力ではあるのですが、その努力で止まってしまっていたことが一番の問題でした。思い当たる部分は本当にたくさんあります。もっとこうすれば便利なんじゃないか、というところは作っていませんでした。ユーザーが何を求めているかなんて考えていませんでした。とにかくクオリティを放棄して最低限のリリースのみをめざしての開発だけを行っていました。それで世の中の成功している人を羨むというのは本当に愚かなことでした。

自分は勝者達とはまだまだ程遠い存在だと思います。でも、Webサービスを成功に導くためにひたすら努力する。この一番大事な道標を、幸いなことに得ることができました。何者でもない、ただひたすらに努力をすることしかできない、小さな一人の人間にとって、それは非常に大きな希望です。この1年の間に知り合って共に歩んできた全ての方々には感謝してもしつくせません。

## 25.6　ひたすら突っ走り始めた

図25.1: Crieit


2018年はガンガンサービスを作ろうということで、引き続き時間を費やしました。その中でCrieitというエンジニア向けの記事投稿サービスを開発しました。[3]

Crieitは今までと違います。サービスの育成に励んでいます。

今まではサービスを作成しても、人が来ないとすぐ放置しました。集客力がないから何をやってもうまくいくわけがない、と諦めていました。サービスのアイデアに魅力がないから人が集まらない。もう何もできることはないと思っていました。

そうではないのです。誰かが使いたいと思ってくれるサービスにするためにひたすら励むことが必要で、徐々にサービスを育成していく必要があったのです。すぐに諦めてしまわずに、目的に向かってちゃんとやることが大切です。そのあたりまえのことを、ようやく意識しながらサービスを運営できています。Crieitは一発バズらせて当てる系統のサービスではないので、今ひたすら育成しています。

もちろん、明らかに誰も、自分すらも使うのが億劫というサービスは厳しいと思います。ちょっと前に「一人もくもく会」というサービスを作ったのですが、だんだんと自分でも何のために使っているのかわからなくなりました。そういうものは改善させるために大きくサービスの道筋を変える必要があると思います。

あともうひとつ違うところとして、最小限の状態で公開する、というところがありました。特にファッションSNSを作った時は、リリース前に必要だと思ったものを全部作りこみました。でも、自分の思い込みで機能を作ってはいけません。どういう機能が必要かは、ユーザー視点で考える必要があります。そのため、Crieitは最小限

の状態で公開しました。ユーザーの声を聞いて徐々に改善したかったのです。中途半端な開発に見えるかもしれませんが、ユーザーと手を取り合うための、最初の一歩でした。

そのおかげか、アドベントカレンダー[4]では、ユーザーが積極的にCrieitに記事を投稿してくれました。

ちなみに現在は並行して色々と他のサービスを作成しています。やはりひとつのサービスだけでは不安ですし、色々なターゲット向けのサービスも作りたいからです。

幸いなことに運営者ギルドにも、マネタイズの会にも、サービスを流行らせている猛者たちがいて、参加しているだけで様々なことを学べます。徐々に自分の考え方もレベルアップしているような気がします（勘違いしているだけの可能性も大です）。

まだ何も形にはできていませんが、2019年にはどこかで渾身の一撃のサービスを生み出したいと考えています。ひたすら頑張って、より大きな幸せを追いかけていきます。


難波聖一 / @dala00

1980年鳥取県生まれ。千葉大学情報画像工学科を卒業後、小さな開発会社で働き、
フリーランスとして独立。在宅・リモートワークにてWebの受託開発を行っている。
プログラミングが趣味のため、普段から個人で色々とサービスを作っている。
「Crieit - プログラマー、クリエイターのためのコミュニティ」を運営。

https://crieit.net


1. Monetize Geek JAMのご案内 https://blog.seekgeeks.net/p/blog-page.html
2. 運営者ギルド https://scrapbox.io/admin-guild-pr/
3. Crieit https://crieit.net
4. 12月1日～25日の間、エンジニアが1日1つテーマに沿った記事を投稿するというWEB上の催し。

# 付録A　141人に聞いた！みんなの個人開発事情

　個人開発者は「個人」で活動しているためか、実態がナゾに包まれています……。そんな個人開発にまつわるアレコレを、赤裸々に回答してもらいました！

・回答人数：141人（すべて任意回答なので、設問によって回答人数が異なります）

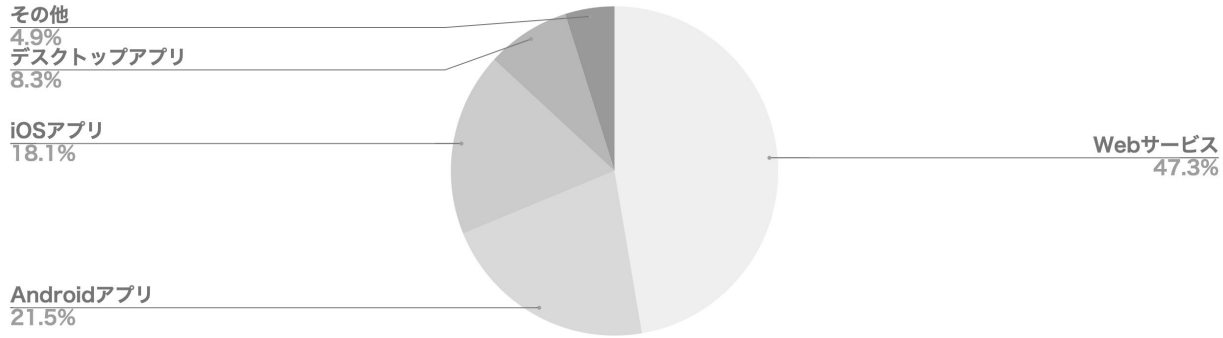
・集計期間：2019年2月5日(火)〜2019年2月15日(金)

・集計方法：Googleフォーム（主にSNSで募集）

**Q1. どんなサービスをつくりましたか？（複数回答可）**

　Webサービスが5割、スマホアプリが約4割、デスクトップアプリが約1割となりました。その他ではChrome ExtentionやSlack App、LINE bot、組み込みシステムなどの回答がありました。

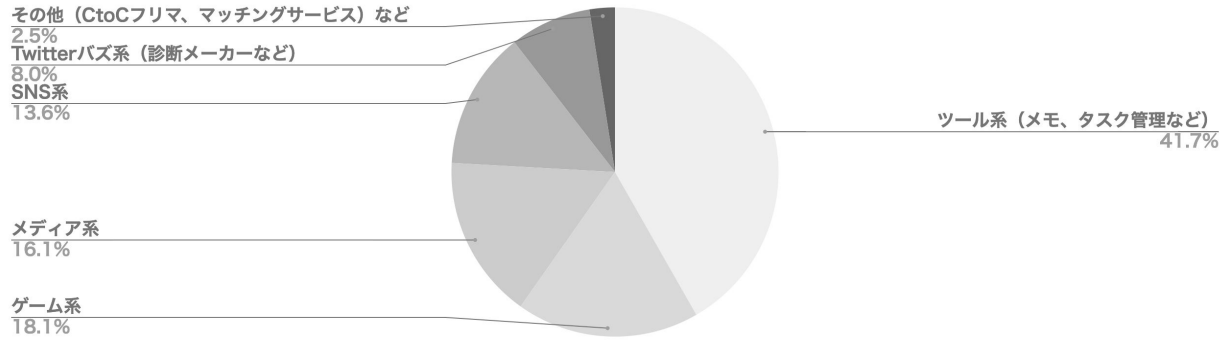
図A.1:




**Q2. つくったサービスの種類はなんですか？（複数回答可）**

　ツール系が最も多く、約半分を占めています。ひとつの機能に特化したツールを作れば開発コストが少なくて済むため、個人開発には向いているのかもしれません。

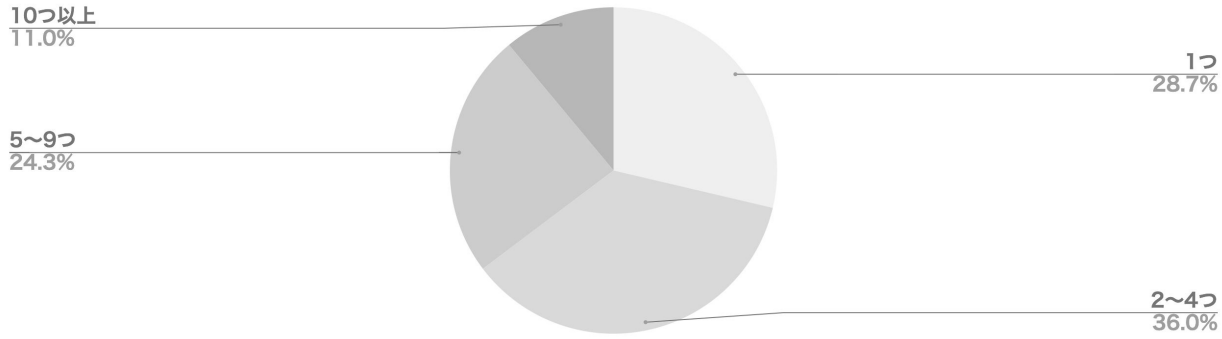
図A.2:



### Q3. いくつサービスをつくりましたか？

ひとつのサービスをじっくり育てるか、次々と新しいサービスを開発するか。開発者によってスタンスが分かれるようです。

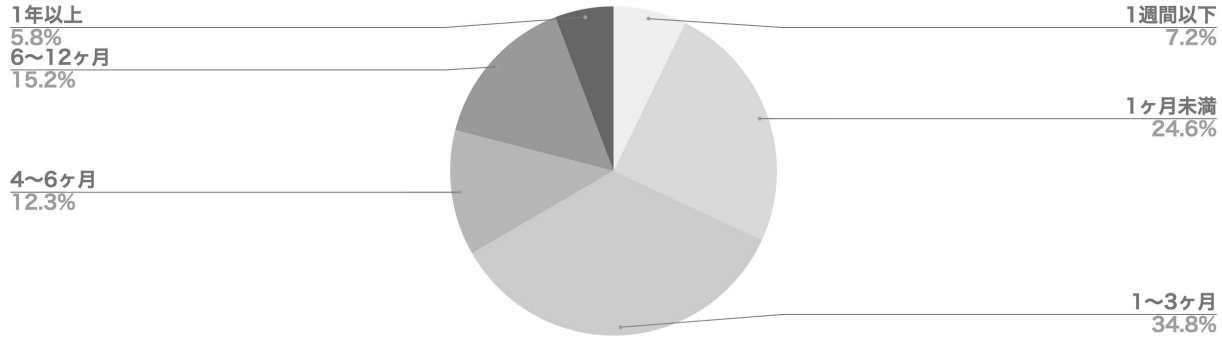
図A.3:




### Q4. サービスのリリースまでにかかった期間はどれくらいですか？

最多は1〜3ヶ月で、次点が1ヶ月でした。スピード感を持って個人開発を進めるのがリリースするためのコツと言えそうです。一方で、半年以上かけて開発しているという人も2割いました。「諦めなければいつか完成する」スタンスでコツコツ開発を続けていくのも大切ですね。

図A.4:




### Q5. 利用技術（言語、フレームワーク、DB、サーバなど）を教えてください

プログラミング言語では、リッチな画面を作れるJavaScript、WEB開発向きのPHPやRuby、iOSアプリに欠かせないSwiftが人気となりました。その他の回答はElixir、Java（Android native）、Perl、Visual Basicがありました。

| | | | | | |
|---|---|---|---|---|---|
| JavaScript | 39 | Java | 14 | Kotlin、Python、TypeScript | 4 |

| | | | | | |
|---|---|---|---|---|---|
| PHP | 33 | C# | 10 | Scala | 3 |
| Ruby | 30 | C++、Golang | 6 | Unity C#、Node.js | 2 |
| Swift | 15 | Objective-C | 5 | その他 | 4 |

フレームワークでは、Webサービスをサクっと作れるRuby on Rails（Ruby）、Vue.js（JavaScript）、Laravel（PHP）が人気のようです。その他の回答にはElectron、Cordova、Ionic、Phoenix、Sinatra、Express、AngularJSがありました。

| | | | | | |
|---|---|---|---|---|---|
| Ruby on Rails | 26 | CakePHP、Django | 7 | Qt、React Native、Riot.js | 2 |
| Vue.js | 24 | Flask | 6 | その他 | 5 |
| Laravel | 20 | Spring Boot | 5 | | |
| Nuxt.js | 9 | Bootstrap、Gatsby | 3 | | |

サーバでは、WEBサービスやモバイルアプリ開発に便利なFirebaseが1位に。AWS、Heroku、VPS（さくら・Conoha・DigitalOcean）といった回答も多かったです。

| | | | | | |
|---|---|---|---|---|---|
| Firebase | 19 | VPS | 16 | その他（S3単体利用、Cloudflare） | 2 |
| AWS | 18 | GCP | 8 | | |
| Heroku | 17 | Netlify | 4 | | |

データベースでは、MySQLが1位（40票）、PostgreSQLとSQLiteが2位（12票）となりました。他にはMariaDB、Redis、MongoDBといった回答がありました。

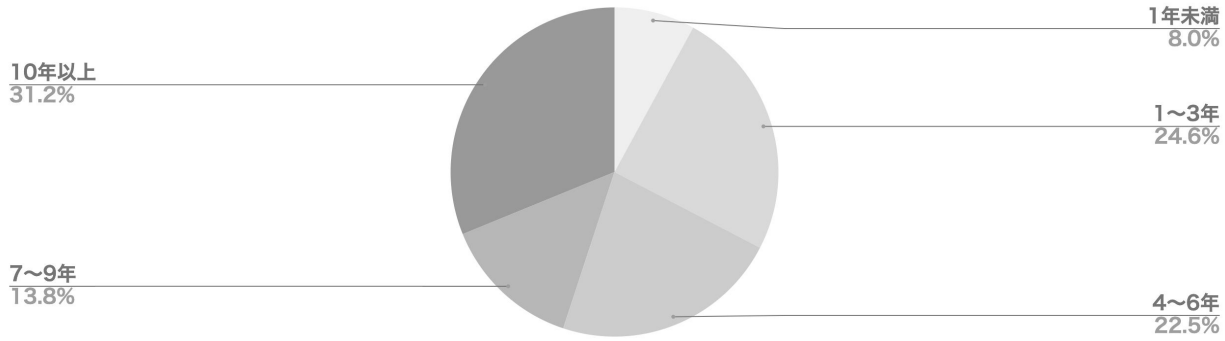
ライブラリーでは、Retrofit、Dagger2、JSoup、RxAndroid、RxJavaなどモバイルアプリ関連の回答が見受けられました。ActiveRecord（RubyのORマッパー）やChainer（機械学習）という回答もありました。

また、CMSを活用する人もいるようです。WordPress、concrete5、Contentful、Tumblrという回答がありました。Gatsbyなどの静的サイトジェネレータも挙がりました。

**Q6. プログラミング歴はどのくらいですか？**

10年以上が3割を占める一方で、プログラミング歴0年～3年の人も3割を占めています。個人開発を始めるのに、「自分にはまだ技術力がないから……」と怖気付く必要はありません！今の自分が作れるサービスをまずは考えてみてはいかがでしょうか？

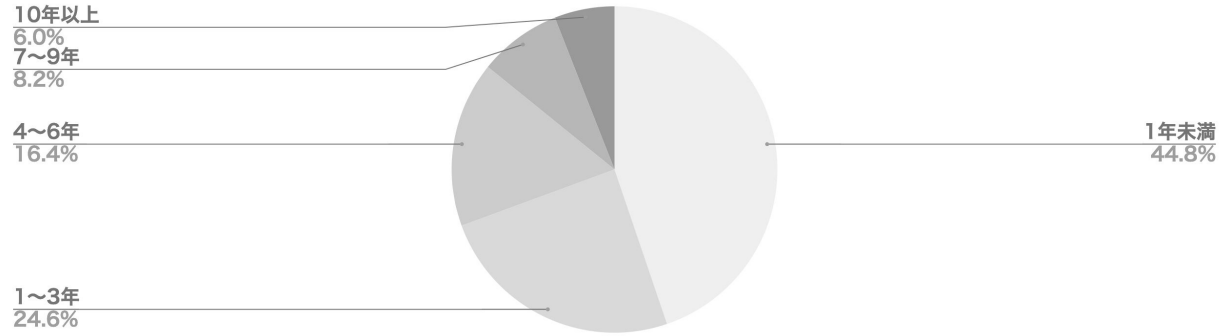
図A.5:




### Q7. 個人開発サービスの運営歴はどのくらいですか？

1年未満が4割以上の一方で、10年以上サービス運営を続けている猛者が5.9%という結果になりました。まだこれからという人も、長く続けている人も、同じように楽しめるのが個人開発の魅力ですね。

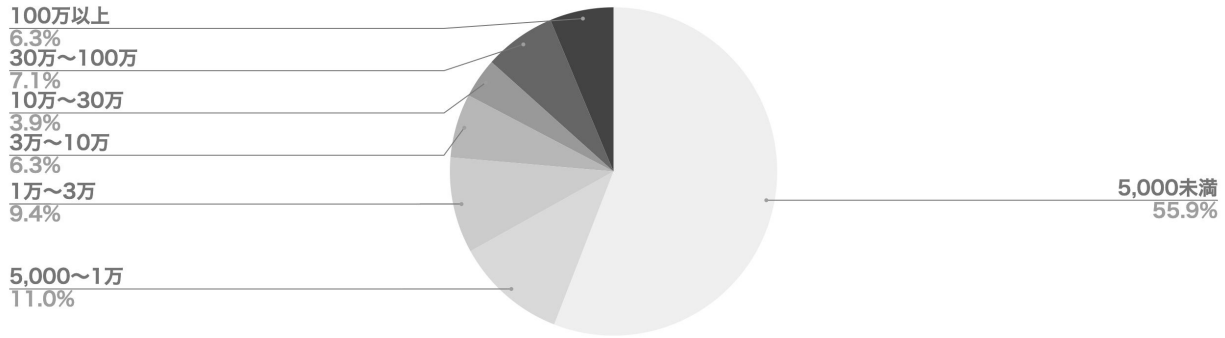
図A.6:




### Q8. 最大の月間PV（ページビュー）はどれくらいですか？

約半分はひっそりと運営されているようです。集客の難しさを体現する結果と言えるかもしれません……。一方で、月間PV100万以上！というツワモノも6%いるとのことでした。

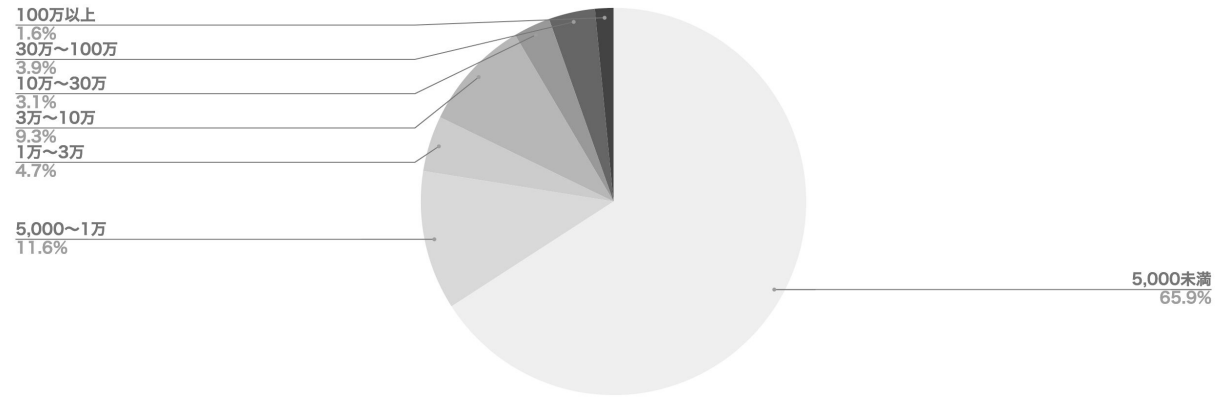
図A.7:



### Q9. 最大の月間UU（ユニークユーザー）はどれくらいですか？

ユニークユーザー数は5000人未満が65%と最多に。個人開発をする場合は、まずは自分の身の回りの人に届けることを考えた方がよさそうです。一方で、回答の1.5%では100万人以上のユーザーに使われているとのことです。そう考えると夢がありますね。

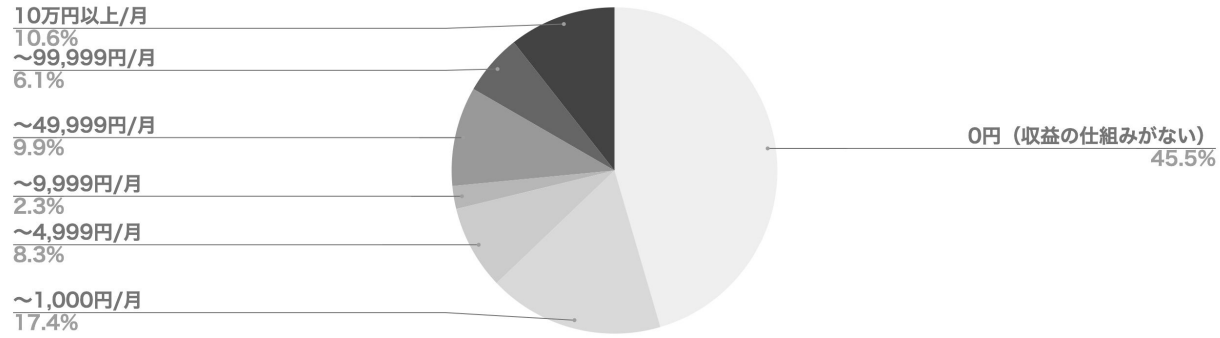
図A.8:




### Q10. 個人開発で月にどれくらいの売上がありますか？

約2人に1人はマネタイズの仕組みを持っていないようです。売上があるケースだと、月1万円未満・1万円以上が半々となっております。ネット上では「○千万稼いだ！」という武勇伝が目立ちますが、過度に期待せず「お小遣い程度でも入ってくればラッキー」と考えるのが大事そうです。ダメ元でも続けていれば、だんだんとお小遣いの規模が増えていくかも……？

図A.9:

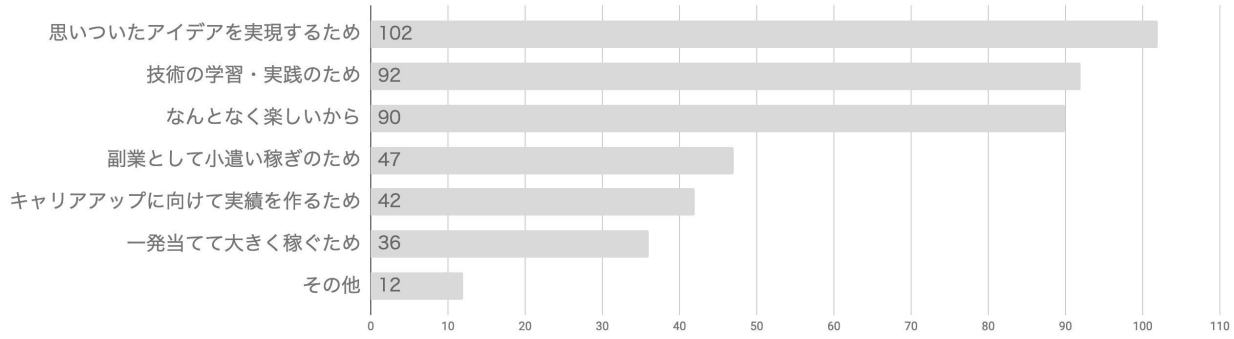



### Q11. 個人開発をはじめた理由はなんですか？（複数回答可）

「こんなサービスがあったら便利なのでは？楽しいのでは？」といったアイデアを実現するために個人開発を始めた人が最多となりました。「技術の学習・実践のため」が第2位、「なんとなく楽しいから」が第3位と僅

差で並んでいます。「その他」では「他の人の役に立ちたいから」「自己実現のため」といった回答がありました。

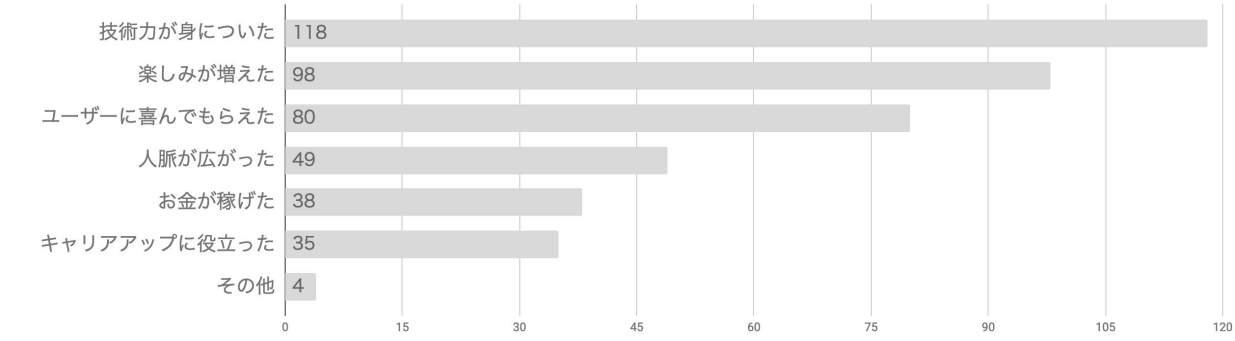
図A.10:




**Q12. 個人開発をやっていてよかったことはありますか？（複数回答可）**

1位は「技術力が身についた」でした。個人開発を始めたときは「自分の技術力を伸ばそう」と意識していなかった人も、開発を続けるうちにスキルが身につくようです。「楽しみが増えた」という人も98名おり、個人開発が毎日の生活に彩りを与えてくれている様子が伺えます！

図A.11:



## あとがき

本書を最後まで読んでいただき、誠にありがとうございます。

## 関係者へのお礼

本書の刊行に当たって、多くの方々にご尽力いただきました。

・インプレスR&Dの山城敬（@kurakake）さんには、本書の出版という壮大な（？）開発プロジェクトに伴走いただきました。

・イラストレーターの湊川あい（@Ilminatoll）さんには、本書のコンセプトをもとに素敵な表紙を描いていただきました。イラストに隠された仕掛けの数々、最高です。

・まごさん、紫苑さん、Maololさんには、編集業務をサポートいただきました。編集アシスタントに応募してくださった他の方々にもこの場を借りてお礼申し上げます。

・執筆者の1人ひぐち（@toiroakr）さんには、快適な執筆・編集ができるようにシステムを構築していただきました。

・25人のクリエイターの皆様には、寄稿＆レビューいただきました。特にSlackやオフラインで日々交流しているメンバーは、一喜一憂を分かち合ったり、相談に乗っていただいたりと、本当にお世話になりました。

・アンケートに回答してくださった141人の皆様には「それぞれの楽しみ方」を読者が考えるためのヒントを提供いただきました。

・同人版書籍を購入・紹介してくださった1,000人以上の読者の皆様には、商業出版のきっかけ・後押しをいただきました。1人1人の顔と名前を知ることはできませんが、1人1人がいたからこそチャンスに巡り会えたのだと感謝しています。

皆様のおかげでこの企画を実現することができました。誰か1人でも欠けたら『個人開発をはじめよう - クリエイター25人の実践エピソード』という書籍は、今とは異なる存在になっていたでしょう。

一方で、多くの関係者を巻き込んだ企画ということで、当初想定を大幅に超えた時間・負担が掛かってしまいました。多くの方々にご迷惑・ご心配をお掛けしたことをお詫び申し上げます。そして、忙しい中でも皆様に前向きにご協力いただけたこと、皆様と力を合わせて出版に漕ぎ着けたことを、嬉しく思います。

## 商業出版に寄せて

本書は、技術同人誌『個人開発がやりたくなる本 - クリエイター13人の実録エッセイ』を元に大幅に加筆・修正したものです。音楽制作で例えると、完パケ音源（Final Mix）版と言えるでしょう。

商業版では、これから個人開発をはじめようとする初学者に向けて「14」のコンテンツを追加しています。同人版の「13」を超える数の原稿を新たに集めました。いずれもクリエイターの背中を押してくれる魅力的なエピソード＆エッセイです。

これほど多くのクリエイターの言葉がつまった「個人開発」の書籍は、過去に存在しなかったと自負しています。

## 同人版との違い

同人版に掲載した2つのコンテンツは、本書に含めていません。

『突然1日100,000PVを達成した地味なサービスの話』と『高専卒の理系Webエンジニアが100社の法人営業を経験した話』をカットしました。インパクトを追い求める開発者にとって学びが多く、読者からも好評のパートでした。

反骨心にあふれた13人の探求者が、試行錯誤しながら自費出版した同人誌です。商業版である本書とは違った読書体験になるでしょう。興味のある方はぜひ同人版もお買い求めください。

## 今を生きるクリエイターに

昨今では不確実性の高まりやテクノロジーの進化に伴って「個」の持つ可能性が注目されつつあります。

250万年前に人類が石器を作ったのも、言ってしまえば個人開発のようなものです。その時からモノ作りの楽しさは変わっていないはずです。

25年前に発売したWindows95が引き金となり、インターネットは個人にとって身近なものとなりました。個人がクリエイティビティを発揮しやすい時代が到来したと言えるでしょう。

こうした時代の節目に、この1冊を、クリエイターである「あなた」に読んでいただけたことを、光栄に思います。こうした時代の節目に、この1冊を、クリエイターである「私たち」が刊行できたことを、光栄に思います。

## 「+1」

本書のキーナンバーである「25」には「24+1」という祈りを込めました。

「24」は、時刻や方位といった形で、人類が歴史の中で慣れ親しんできた数字です。しかし、時計や方位磁針に従うだけでは、辿り着けない場所があります。1人1人が心のままに進むことでしか、辿り着けない場所があります。

250万年前に石器を作ったクリエイターがきっとそうであったように。25年前にWEBサイトを作ったクリエイターがきっとそうであったように。自由な心でクリエイティビティを発揮するからこそ、辿り着ける場所があるはずです。

それこそが「個」の可能性だと思っています。自由気ままな風に吹かれて、未開の地へと旅立つような「+1」を後押ししたい。読者のこれまでの生き方、24時間の過ごし方から、1歩を踏み出せるように後押ししたい。本書がそのような1冊であってほしい。そう願っています。

## 個人開発をはじめよう！

あなたにとっての「+1」は何でしたか。

どのクリエイターの言葉が、心に響きましたか。どのクリエイターの言葉が、マネしたくなりましたか。どのクリエイターの言葉が、モチベーションに繋がりましたか。

あなたがクリエイターとしての一歩を踏み出したくなるような「何か」と出会えたのであれば幸いです。

ぜひ個人開発を楽しんでください。心のままに。

（発行代表・企画者 @yuzutas0）

編者紹介

## ゆずたそ

（自称）企画屋・コンセプトデザイナー。WEBに興味を持ったきっかけは、依頼を受けて小学校の特別授業を企画したこと。グラミン銀行と提携してSkypeによるオンライン国際交流を実現。子供達の笑顔が国境を超えた瞬間を目の当たりにして、ITの可能性に気付く。

◎本書スタッフ
アートディレクター/装丁：岡田章志＋GY
編集協力：飯嶋玲子
デジタル編集：栗原 翔

〈表紙イラスト〉
湊川 あい（みなとがわ あい）
フリーランスのWebデザイナー・漫画家・イラストレーター。マンガと図解で、技術をわかりやすく伝えることが好き。著書『わかばちゃんと学ぶ Webサイト制作の基本』『わかばちゃんと学ぶ Git使い方入門』『わかばちゃんと学ぶ Googleアナリティクス』が全国の書店にて発売中のほか、動画学習サービスSchooにてGit入門授業の講師も担当。マンガでわかるGit・マンガでわかるDocker・マンガでわかるUnityといった分野横断的なコンテンツを展開している。
Webサイト：マンガでわかるWebデザイン http://webdesign-manga.com/
Twitter：@llminatoll

**技術の泉シリーズ・刊行によせて**

技術者の知見のアウトプットである技術同人誌は、急速に認知度を高めています。インプレスR&Dは国内最大級の即売会「技術書典」（https://techbookfest.org/）で頒布された技術同人誌を底本とした商業書籍を2016年より刊行し、これらを中心とした『技術書典シリーズ』を展開してきました。2019年4月、より幅広い技術同人誌を対象とし、最新の知見を発信するために『技術の泉シリーズ』へリニューアルしました。今後は「技術書典」をはじめとした各種即売会や、勉強会・LT会などで頒布された技術同人誌を底本とした商業書籍を刊行し、技術同人誌の普及と発展に貢献することを目指します。エンジニアの"知の結晶"である技術同人誌の世界に、より多くの方が触れていただくきっかけになれば幸いです。

株式会社インプレスR&D
技術の泉シリーズ　編集長　山城 敬

**●お断り**
掲載したURLは2020年1月1日現在のものです。サイトの都合で変更されることがあります。また、電子版ではURLにハイパーリンクを設定していますが、端末やビューアー、リンク先のファイルタイプによっては表示されないことがあります。あらかじめご了承ください。

**●本書の内容についてのお問い合わせ先**

株式会社インプレスR&D　メール窓口
np-info@impress.co.jp

件名に「『本書名』問い合わせ係」と明記してお送りください。
電話やFAX、郵便でのご質問にはお答えできません。返信までには、しばらくお時間をいただく場合があります。
なお、本書の範囲を超えるご質問にはお答えしかねますので、あらかじめご了承ください。
また、本書の内容についてはNextPublishingオフィシャルWebサイトにて情報を公開しております。
https://nextpublishing.jp/

技術の泉シリーズ

# 個人開発をはじめよう！クリエイター25人の実践エピソード

2020年4月3日　初版発行Ver.1.0（リフロー版）

編　者　ゆずたそ
編集人　山城 敬
発行人　井芹 昌信
発　行　株式会社インプレスR&D
〒101-0051
東京都千代田区神田神保町一丁目105番地
https://nextpublishing.jp/


ISBN978-4-8443-7814-3


NextPublishing®

◉本書はNextPublishingメソッドによって発行されています。
NextPublishingメソッドは株式会社インプレスR&Dが開発した、電子書籍と印刷書籍を同時発行できるデジタルファースト型の新出版方式です。 https://nextpublishing.jp/