2023
秋号

百萬石

Hyakumangoku Vol.61

Published by MAAA

部長挨拶

gae

gae@mma.club.uec.ac.jp

「百萬石」をお手に取っていただき，ありがとうございます．弊誌 (以下，部誌) は，電気通信大学の公認サークル MMA が新歓期 (春号) と弊学が開催する調布祭時期 (秋号) の年 2 回発行・頒布しているものです．MMA とは，”Microcomputer Making Association”の頭文字に由来しており，1975 年に創立されたサークルです．現在では計算機に関連することを中心に，ネットワークやセキュリティなどの分野でも活動しています．部誌には，MMA の部員が各々に行なっている活動を記事にしたものが掲載されています．内容は多岐に渡り，高度なプログラミングの話から旅行記など，MMA の活動の自由さが感じられるものとなっております．

さて，この「秋号」は調布祭に頒布されるものです．昨年度に引き続き，今年度も MMA は調布祭に対面で参加するため，ありがたいことに直接部誌をお渡しできます．オンライン参加時代に作った，これまでの歴代の部誌がすべて pdf で読めるようになっている，部員が作成したサイトもあります．先輩方の歴史とともにお楽しみいただければと思います．

今回は秋号ということで，高校生も多く読むでしょう．そこで，電通大志望の高校生向けに少し話しましょう．電通大は，実は国公立なんですね．高校生の皆さん知ってました？なので，一般受験では共通テストと 2 次試験を受ける必要があります．2025 年からはそのどちらにも情報 (2 次は選択) が追加されます．さすが電通大ですね．

さあ，そんな中で MMA です．電気通信大学の中で，さらにプログラミングやサーバーを扱いたいという計算機だいすきといった人たちももちろんいます．が，毎年初心者が数多く入部しています．今年度の一年生もほとんどが初心者です．高校時代からパソコンに触れて慣れておくのももちろんよし．ですが，大学に入ってからしっかりと学んでも全く遅くはありません．MMA に入って勉強すればいいんだもんね．真面目に，大学の授業でも，1 年生の時にコンピュータの基礎からかなりのボリュームを学びます．2 年生以降でも，配属によってプログラミングの演習をかなり行うところもあります．大学から始めても全然大丈夫です．

MMA は**計算機 (コンピュータ) を使っていれば，活動内容は完全に自由です．**部内でもさまざまな活動が行われているため，いま誰が何をしているのか私も把握していません．わからないことがあっても質問すれば誰かしらが答えてくれます．私も電通大，そして MMA に入ってからプログラミングを始めた 1 人です．そんな人がいま部長をできているのだから MMA は勉強にとてもいい環境が揃っており，それを十分に活かすことができれば，プログラミングをしっかりと学び楽しむことができるでしょう．

MMA の宣伝はこの辺にして，MMA の近況です．9/16 ～ 9/18 の 3 日間，4 年ぶりに MMA は伊豆に合宿に行ってきました．「MMA の合宿なんて，どうせプログラミング三昧なんでしょ？」と思ったそこのあなた．全く違います!!!MMA の合宿は**花火合宿**です．線香花火でも手持ち花火でもなく，打ち上げ花火です．1 日目は移動してゆっくりする．2 日目は花火の準備と打ち上げ本番．3 日目は帰る．といった感じに花火合宿でしたね．部員に花火の免許を取ってもらい，関係各所に連絡をしてやっと打ち上げられました．打ち上げはなんと 7 年ぶり．雨が降ったり，コロナが流行ったりとなかなか打ち上げられませんでしたが，今年やっと打ち上げることができました．



最後になりましたが，MMA は**自分のやりたいことを実現できる場**です．充実した環境，助けになる先輩方，こんな場所で活動する部員たちの部誌を読んでみてください．難しいものもあるとは思いますが，きっと面白いものもあります．ぜひご一読ください．

2023 年 11 月吉日

百萬石 Vol.61

Hyakumangoku Vol.61

2023 Autumn

MMA

目次

第 1 章

Entersys 作成記

gae

1.1 はじめに

こんにちは．私の記事を読んでくださりありがとうございます．2023 年 MMA 部長の gae です．部長挨拶以外で部誌を書くのは初めてですが，今回は気まぐれに作りたいものを作る私の製作物 Entersys についてお話しさせていただこうと思います．本題とは関係ありませんが，ぜひ本誌巻頭の部長挨拶も読んでいただけるとありがたいです．では本題に参りましょう．

1.2 Entersys と作成動機

Entersys とは，一言で言うと MMA の部室の入退室管理システムです．コロナが流行ってきてから，部室に入退室する人や，だれが部室にいるのかを把握する必要が出てきました．手打ちで入力していたのを自動化しようということになり，Entersys が生まれました．

1.3 Entersys の構成

Entersys の構成は章末の図のようになっています．まず，学生証についているバーコードから学籍番号を読み取り，標準入力にて番号を入力します．その学籍番号をもとに入部届の学籍番号から MMAid を取得します．その後，MMAid が見つかれば，在室者リストに追加，部室にすでにいる場合は削除します．その結果を Slack にも送信します．基本的な機能はこれだけです．簡単ですね．おまけとして，毎日入部届から名簿を更新したり，Slack のアイコンを更新したり，退室処理を忘れた部員をリストから定期的に削除する機能もあります．

1.4 作成にあたって

作成には，Python と Google Apps Script を用いました．Python は，学籍番号の読み取り，学籍番号から MMAid の取得，Slack への通知，部内リストの更新，退室処理の忘れた部員の削除を行っています．Google Apps Script は，アイコンの更新，毎日入部届から名簿を更新する処理を行っています．

Entersys の作成にあたって初めて Python を使いましたが，様々なモジュールがあるため，実装したい機能がある時にすぐに作成することができました．

1.4.1 Slack の API

Slack の API を使うことで，Slack にメッセージを送信することができます．Slack の API は，Webhook というものを使うことで，簡単にメッセージを送信することができます．Webhook とは，Slack の特定のチャンネルにメッセージを送信するためのものです．POST リクエストを送信することで，Slack にメッセージを送信することができます．また，この機能では，送信するメッセージ内容の指定だけでなく，送信する bot のアイコンや名前，その他文字の装飾も指定することもできます．下のプログラムは，Webhook を用いて Slack にメッセージを送信するプログラムです．送信するユーザの名前とアイコンを取得し，送信メッセージを指定して，自動的に Slack にメッセージを送信することができます．これにより視覚的に誰が部室に来たのか，部室から出るのか，部室にいるのかを確認することができます．これを用いた結果，図 1.4 のようにメッセージを送信することができます．

```
# Slack通知処理
user_icon = slackicon.get_url(id) # アイコンの取得
slack_user_icon = slackicon.get_url(id) # アイコンの取得
blocks = [ # 通知用のブロックを作成
    {
        "type": "context",
        "elements": [
            {"type": "mrkdwn", "text": f":{message}:  在室"},
        ],
    }
]

for i in range(len(inside)): # slackの画像表示幅に合わせて改行
    user_icon = slackicon.get_url(inside[i])
    if i % 8 == 0 and i != 0:
        blocks.append({"type": "context", "elements": []})
    if user_icon:
        blocks[i // 8]["elements"].append(
            {
                "type": "image",
                "image_url": user_icon,
                "alt_text": f"@{inside[i]}",
            }
        )
    else:
        blocks[i // 8]["elements"].append(
            {"type": "mrkdwn", "text": f"@{inside[i]}"}
        )

# 通知を送信
requests.post(
    WEB_HOOK_URL,
    data=json.dumps(
        {
            "channel": "#room_io", # 送信先のチャンネル
            "username": id, # 送信者の名前
            "icon_url": slack_user_icon, # 送信者のアイコン
```

```
            "blocks": json.dumps(blocks), # 送信するメッセージ
        }
        )
    ),
```

1.4.2 学籍番号と MMAid の対応

初めに作った試作品では，毎回入部届のフォームのスプレッドシートから参照するようにしていたのですが，これだと処理が遅くなってしまいます．そこで，入部届のスプレッドシートから MMAid と学籍番号の対応表を作成し，それを参照するようにしました．これにより，処理が高速化されました．また，毎日入部届のスプレッドシートから MMAid と学籍番号の対応表を更新するようにしています．

```
import csv
import update_member
csvfile = "部員名簿ファイルのパス"

def get_id(number):
    id = ""
    length = len(number)

    # 学籍番号の桁数が7桁,バーコードから読み取ると再発行回数も含まれるので7桁にする
    if length == 7 or length == 8:
        if length == 8:
            number = str(int(number) // 10)

        # 名簿ファイルから検索
        with open(csvfile) as f:
            reader = csv.reader(f)
            row_list = [row for row in reader]
        lastrow = len(row_list)
        for i in range(0, lastrow):
            if number == row_list[i][1]:
                id = row_list[i][0]
                return id

        # 見つからなかった時入部届から部員名簿の更新
        update_member.update_member()

        # もう一度検索
        with open(csvfile) as f:
            reader = csv.reader(f)
            row_list = [row for row in reader]
        lastrow = len(row_list)
        for k in range(0, lastrow):
            if number == row_list[k][1]:
                id = row_list[k][0]
                return id
```

1.4.3 入退室判定

入室したのか退室したのかを判定するためには，部室にいる人のリストが必要です．そこで，Entersys が途中で落ちてしまっても大丈夫なように，配列に格納するのではなく，テキストファイルに保存するよ

うにしました．次のプログラムは，main 関数内の入退室に関する部分です．Python 標準のライブラリを用いて，テキストファイルから入室者リストを読み込み，そのリストに学籍番号があれば退室処理を行い，なければ入室処理を行います．処理完了後は，テキストファイルに入室者リストを書き込みます．また，中にある `message`は，slack 送信する際の図 1.4 の矢印の向きの判定に使われています．

```
message = "enter"
with open("./../data/inside.txt", "r", encoding="utf-8") as f:
    inside = f.readlines()
inside = [line.rstrip("\n") for line in inside]
for one in inside:
    if one == id:
        inside.remove(id)
        message = "exit"
        break
if message == "enter":
    inside.append(id)
```

1.4.4 細かな機能

1. **退室処理**　退室処理を忘れた部員を毎日削除するために，`schedule`モジュールを使って定期実行するようにしています．入室者リストをテキストファイルで管理しているので，毎朝そのファイルを空にしています．
2. **slack アイコンの更新**　`schedule`モジュールを使って定期実行するようにしています．図 1.4 のように slack のアイコンで誰が部室にいるのかを表示しています．このアイコンを Google Apps Script で管理しています．
3. **並列実行**　学籍番号の読み取りとアイコン更新などの定期実行を同時に行わないとならないため，`threading`モジュールを使ってプログラムの並列実行を行っています．

1.5 まとめ

Entersys はこの記事が短いように，実際にやっていることもかかった日数的にもさほど難しいものではありませんでした．実際に，Entersys が正しく動作するまでにかかった日数はおよそ 2 日，その後 slack 送信部分の装飾に 1 週間，細かな調整だったり機能を追加するのに 1 週間ほど．計 2 週間ほどで本格的に運用を開始することができました．

私は今回初めて Python に触れましたが，言語的にも作成内容的にも，初心者が挑戦する初めてのプログラムとしてはちょうどいいものだと思いました．私は競プロなどコンテストに出るよりも，「何か不便なものを解消するために何かを作る」ということをモチベに活動をしているので，知識としてはまだまだですが，今回のような作成の中でプログラミングを学習しています．不便解消と勉強を兼ねて，今後も気まぐれに「めんどくさいなー」だったり「こんなのあったらいいなー」を実現していきたいと思います．

図 1.1 バーコードリーダ

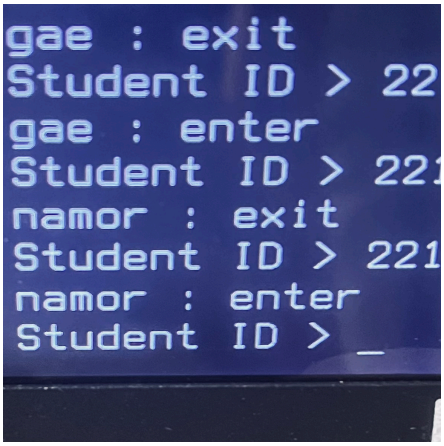


図 1.2 Entersys 出力画面

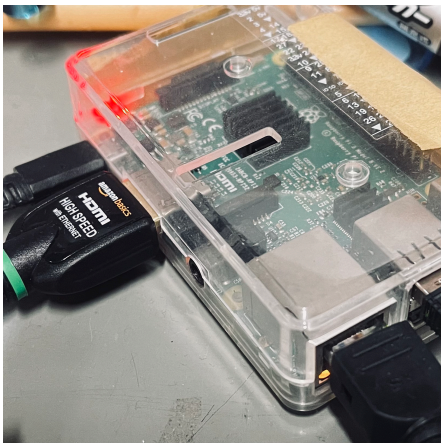


図 1.3 Entersys 動作用ラズパイ

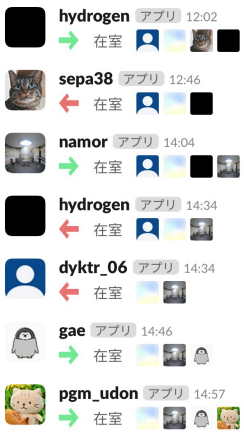


図 1.4 Slack 通知部分

第 2 章

友達が例大祭出るらしいから LaTeX で手伝ってみた

Udon

2.1 はじめに

毎度お疲れ様です。Udon です。

もう秋ですね。秋と言えば秋例大祭。というわけで今回は表題の通り「友達が例大祭に出るらしいから技術提供してみたぜ！」というイケイケ () な内容を書いていこうと思います。

2.2 例大祭ってなんだ

例大祭っていうだけだと結構一般名詞な感じで、神社が定期的に開催するお祭りという意味です。ただ、ここでいう「例大祭」とは、「博麗神社例大祭」のことです。

詳しい解説は電電〇通信。とかに譲るとして、ここでは簡単な説明にとどめますが、要するに東方 Project 作品オンリーの同人誌即売会、簡単に言うと東方限定のコミケという感じです。

同人誌即売会といいつつ取引される商品は多様で、音楽 CD やゲームの ROM、小説や漫画などが作り手から買い手に直接売られるという形でイベントが進行していきます。

作り手はファンの顔を直接見ることができ、ファンは憧れのアーティストに会えたり通常より安く[*1]、そして最速で新作を買うことができる、という素晴らしいイベントになっています。

自分は高校生のころからちょくちょく参加しています。今までは買い手として参加してきたのですが、今回は友達が個人サークルを出すということで、売る手伝いという名目で、売り手側として参加することになりました。

で、せっかく売り手側で参加するなら売り子以外でもなんかしたいな、ということで、友達が出品する小説のレイアウト成型を大学生になってから学んできた LaTeX の知識を用いてやってみようと思い立ちました。今回の部誌ではそのことを書いていこうと思います。

[*1] 卸売業者が絡まないから

2.3 で、具体的にどんなことしたの？

本文は全て友達が書くことになっていたので、自分がやったことはもらった本文を校閲し、作成した TeX ファイルの所定の位置に入れ込み、レイアウトを指示に従って調整するという作業でした。加えて、あとがきを少し書かせてもらったりもしました。

細かく話をすると、今回やってみたかったのは以下のものです。

- 縦書きの実現
- ノドなど、細かいレイアウトの調整
- 同人誌に適したフォントの採用
- あとがきの成型

この四つをどうにかこうにか実現してみようという趣旨で開発をしました。

2.4 ソースコード

さて、毎度おなじみソースコードのお時間です。参考文献に列挙したサイトの内容を参考にさせていただいて、以下のファイルを作成しました。

```
\documentclass[10pt,dvipdfmx,titlepage]{copybook}
\usepackage[papersize={111mm, 154mm}]{geometry}
\usepackage{here}
\usepackage[dvipdfmx]{graphicx}
\usepackage[dvipdfmx]{color}
\usepackage{url}
\usepackage[deluxe]{otf}
\usepackage[noalphabet, unicode]{pxchfon}
\setminchofont[0]{GenEiKoburiMin6-R.ttf}
\setgothicfont[0]{GenEiMGothic2-Black.ttf}
\usepackage{midpage}
\usepackage{lmodern}
\usepackage[T1]{fontenc}
\setlength{\columnsep}{2zw}
\usepackage[Q=11,H=21,W=42,L=15,footskip=10mm,jisfontzoom,ten=15mm,nodo=20mm,tate]{hanmen}
\usepackage{fancyhdr}
\setlength{\headwidth}{\textheight}
\pagestyle{fancy}
\cfoot{\thepage}
\renewcommand{\headrulewidth}{0pt}
\makeatletter
\chardef\nvlsty@zenkakuSpace=\jis"2121\relax
\newcommand{\tcy}[1]{\relax
    \nvlsty@zenkakuSpace\kern-1zw\relax
    \leavevmode\hbox to 1zw{\relax
        \centering\rensuji*{#1}\relax
    }\relax
    \kern-1zw\nvlsty@zenkakuSpace
}
\makeatother
```

```
    \begin{document}
    \thispagestyle{empty}
    \switchwritingdirection
    \begin{midpage}
        \begin{huge}
            \begin{center}
                %タイトル
            \end{center}
        \end{huge}
    \end{midpage}
    \clearpage
    \switchwritingdirection
    \thispagestyle{empty}
        {\mgfamily{
            \begin{Large}
            \begin{center}
                %目次
            \end{center}
            \end{Large}
        }}
    \clearpage
    \thispagestyle{empty}
    \begin{midpage}
        \begin{LARGE}
            \begin{center}
                %章の名前
            \end{center}
        \end{LARGE}
    \end{midpage}
    \clearpage

    %ここに本文

    \clearpage
    \switchwritingdirection
    \section*{あとがき}
    %あとがきを書く
    \vspace*{\stretch{1}}\\
    \hrulefill
    \begin{flushright}
        \begin{minipage}{0.8\hsize}
            \begin{description}
                \item \begin{Large}\textbf{タイトル}\end{Large}
                \item{発行日:}
                \item{作者:}
                \item{表紙:}
                \item{表紙デザイン:}
                \item{印刷:}
                \item{連絡先:}
            \end{description}
        \end{minipage}
    \end{flushright}
    \hrulefill
    \end{document}
```

ソースコード 2.1　main.tex

このほかにドキュメントクラスの設定として「copybook.cls」というファイルを用いていますが、長くな

るので割愛します。どうやら tbook.cls を改造したものらしいです。

ビルドすると以下のようなレイアウトになります。ちなみにビルドには platex を用いています。

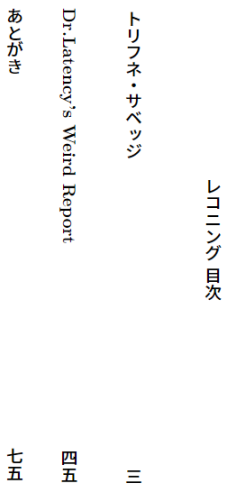


図 2.1 目次

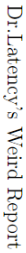


図 2.2 タイトル

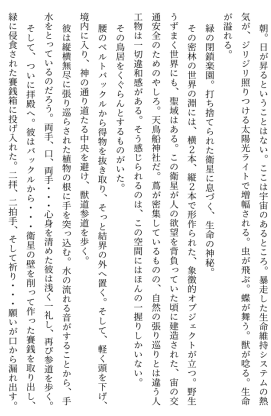
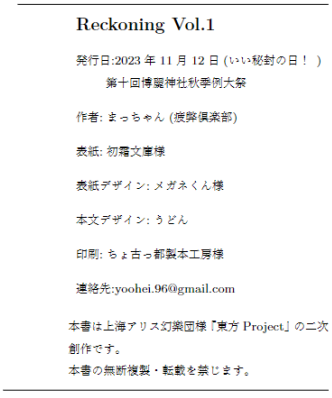


図 2.3 本文



Reckoning Vol.1

発行日:2023 年 11 月 12 日 (いい秘封の日！)
第十回博麗神社秋季例大祭

作者: まっちゃん (疫弊倶楽部)

表紙: 初霜文庫様

表紙デザイン: メガネくん様

本文デザイン: うどん

印刷: ちょ古っ都製本工房様

連絡先:yoohei.96@gmail.com

本書は上海アリス幻樂団様「東方 Project」の二次創作です。
本書の無断複製・転載を禁じます。

図 2.4 奥付

さて、ここから細かい部分を解説していこうと思います。

2.4.1 レイアウト

基本的には copybook.cls を用いていますが、これは LaTex 標準の tbook.cls にちょっと手を加えたくらいなので割愛します。

紙のサイズは A6 でと言われたので直接指定しています。横 111mm 縦 154mm ですね。文字の大きさは 10 ポイントにしています。

レイアウトに関しては九州大学文芸部なるものが作った hanmen パッケージを使わせていただきました。これを用いるとめちゃくちゃ簡単にレイアウトのパラメータを設定することができるらしいです。

設定部分は以下のような感じになっています。

```
\usepackage[Q=11,H=21,W=42,L=15,footskip=10mm,jisfontzoom,ten=15mm,nodo=20mm,tate]{
hanmen}
```

Q っていうのは級のことで、文字サイズらしいです。ポイントと同じなのかよくわからなかったのですが、11 にしたらいい感じになったので 11 にしました。H は歯で、行送りの単位になっています。これは作者の指示で 21 に設定しました。また、W と L はそれぞれ行の長さと行の数を指定しています。これも 42 字 15 行という指示のもとに設定しました。

その後、フッタの位置を fookskip オプションで指定しました。10mm となっていますが、確かページの一番下からの距離だったと思います。jisfontzoom というのは文字サイズの調整を行うもので、jarticle.cls や tbook.cls を用いる際に必要になります。今回は後者を改造したクラスファイルを用いているので指定が必要です。

最後に、天とノドの距離を指定します。直接距離で指定するわけですが、既存の値に加算したりすることもできるらしいです。今回は直接していしました。天を 15mm、ノドを 20mm にしています。偶数ページと奇数ページでノドの位置が自動で変わるようになっています。すごい。

最後に、tate オプションを付けます。これによって縦組みになります。本当にこれだけでいいのかってくらい簡単に縦組みになるわけですね。

2.4.2 フォントの変更

小説同人誌ではフォントが大事になります。LaTeX ではデフォルトでセクション名にゴシック体、それ以外に明朝体、みたいな感じでフォントの割り当てが行われているらしいです。要はこういったフォントを好きなものに変えてみたい、というわけです。

調べていたらフォントマップを埋め込んだりする方法があったのですが、どうにも難しそうで簡単にできる方法はないものかと探していたら、pxchfon パッケージを用いる方法がありました。

記法は以下のような感じになっています。

```
\usepackage[deluxe]{otf}
\usepackage[noalphabet, unicode]{pxchfon}
\setminchofont[0]{GenEiKoburiMin6-R.ttf}
\setgothicfont[0]{GenEiMGothic2-Black.ttf}
```

まず、otf パッケージを使用して OpenType フォントを利用できるようにしました。次に pxchfon パッケージを導入しています。オプションがついていますが、noalphabet はラテン文字のフォントを変更しないという意味で、unicode はエンコードに関するオプションとなっています。

でもって、次にフォントを指定するわけです。setminchofont や setgothicfont を用い、直接フォントを指定します。フォント名の指定は ttf ファイルの名前で行えばいいわけなんですが、ここで重要となるのはフォントファイルの置き場です。当然 tex が見える位置にフォントファイルを置く必要があるのですが、今回は `C:\texlive\texmf-local\fonts\source` にフォントファイルを配置しています。用いたフォントは源瑛フォントというフリーフォントです。DL して利用させていただきました。

でもって、指定したフォントを本文で使っていくわけなのですが、明朝に設定したほうは特に何もしなくても本文のフォントに反映されます。ゴシックに設定したやつは mgfamily という命令を書くと、その範囲内でフォントが反映されるようになっています。具体的には、ソースコードの 44 行目から 50 行目あたりを参照してみてください。

他にも細かくフォントの種類があり、それぞれにフォントを個別に割り当てることができるらしいのですが、今回はフォントを二種類しか使わないとのことだったのでこの 2 つだけ割り当てました。

2.4.3 あとがきの成型

あとがき部分は横書きになっています。これは switchwritingdirection を使えばよく、この命令をした後は文書全体に縦書き設定がしてあっても横書きにできる、という仕様になっています。あとは普通に書いていけば大丈夫です。

奥付はスペースと水平線で区切り、右寄せしたミニページを作って、description 環境を用いて実現しました。大きくするテキストは適宜サイズ変更命令を挟みました。

各作品のタイトルページや目次なども midpage などを用いていい感じに成型しました。これはテキストを紙面の中央に寄せてくれるという命令ですね。

2.5 実際に発行したもの

11 月 12 日の博麗神社秋季例大祭で実際に発行したものが以下のものになります。

図 2.5　実際に発行したもの

わかる人にはわかると思いますが、秘封ですね。秘封を MARVEL 的に解釈したやつと、昔書いたテーマのリメイクって感じの二本立てらしいです (作者談)。表紙デザインや印刷はそれぞれ別の人に担当していただきました。

売れ行きは結構好調でした。35 部くらい刷って、ほとんど売れていった感じですね。まぁ通りかかった人全員が見てくれるわけでもなく、見たとしても買わずに行ってしまったり (自分もよくやります)、一緒に売ってあった家系ラーメンの小説だけ買って行ったり (インパクトがすごいので) していました。その分自分が関わった小説を買っていただけた時は嬉しかったですね。

2.6　おわりに

今回は同人小説を LaTeX を使ってイケイケに書いてみよう！という内容でした。何もわからない状態から組み立てるのは結構大変でしたが、空白の設定とか、段組みの設定とかは慣れていくうちにけっこうできた、って感じでした。

同人小説の醍醐味ともいえるフォントの変更ですが、調べてみたら結構あっさりできる感じでした。正直フォントの導入が一番心配だったのですが、無事にできてよかったです。

本文を書いた友達は今まで Word でやっていたらしいですが、これからは今回作成したテンプレート

に表紙とか本文を入れるだけで成型ができるので、少しは作業が簡単になるかなと思います。

最後に雑談します。実はこれ締切日過ぎてるんですよね。部誌担当が締切落としてどうするねん、って感じなんですが、本当に申し訳ない。一応言い訳をさせてもらうと、このコードの成型ができたのが 11/5、例大祭が 11/12 で、締切が 11/14 です。現物が手に入ったのが 11/12 だったので、かなりスケジュール的にきつかった感じですね[*2]。例大祭前日にコーヒー飲みすぎてカフェイン中毒になって体調崩したり、何かいろいろ大変でしたが、何とか部誌書けて良かったと思っています。

ではまた春部誌でお会いしましょう。というか、この後 LT のスライドとかアドベントカレンダーとか書かなきゃなぁ…。

参考文献

[1] https://reitaisai.com/
[2] https://qiita.com/zr_tex8r/items/a13c195d42b7fca69378
[3] https://qdaibungei.github.io/latex/documents/2017-02-18-hanmen/
[4] https://okoneya.jp/font/genei-koburimin.html
[5] https://qiita.com/makisyu/items/4ee7e0269a711c9db21c
[6] https://note.com/asagiripenguin/n/nbd5cbc19ff15

[*2] 現物の写真以外の所を前もって書いとけ案件でしかない

第 3 章

個人用日記アプリ

namor

3.1 はじめに

今年度は新歓, 会計, 合宿, 花火を担当しました namor です. 合宿では花火も打ち上げに成功し, 事故なく終われたので良かったです. そんな今回の記事は, 初めての Web アプリ作りのお話です. 夏休みからゆっくり作っていてまだ完成していませんが経過報告をしていきたいと思います. そして何か改善点があれば教えて頂きたいです.

3.2 制作動機

これまで, Google Form を用いて日記を書いてスプレッドシートをデーターベースとして使っていました. ですが, あとから内容の修正がシートからでしかできない, 内容が増えるほどシートが大きくなり見返すときに重くなるなど欠点が多々ありました. 写真を見るのもセル内挿入では画質が荒いので, リンク先に飛んでドライブに入っているファイルを開かなければなりません (そもそも圧縮されて保存されます). そこで今回勉強がてら日記の投稿, 検索ができ高画質な写真を残せる Web アプリを作りました.

3.3 主な仕様

- Flask(Python の Web フレームワーク)
- sqlite(データベース)
- bootstrap(CSS のデザインテンプレとして利用)

3.4 このアプリの構造

3.4.1 ディレクトリ

このアプリは主に 3 つで構成されていて Flask 公式チュートリアルを参考に作りました. 図 3.1 は簡易的なディレクトリ構造です.

auth

認証関連の機能を提供しています. 主にユーザー認証や新規登録に関する機能があります.

crud

CRUD（Create, Read, Update, Delete）操作に関連するアプリです. 主にユーザーデータの作成, 編集, 表示などの基本的な機能を提供します.

diary

日記に関連するアプリケーションで, コンテンツの表示や編集を担当します. メインのページデザインは diary 内の base.html で行っています. ユーザーはこのアプリを使用して日記を投稿し, 投稿した画像はユーザごとに images 内にフォルダを作成し, そこに画像が保存されるようになっています.

```
.
├── apps
│   ├── auth
│   │   ├── templates
│   │   │   └── ~.html
│   │   ├── forms.py
│   │   └── views.py
│   ├── crud
│   │   ├── templates
│   │   │   └── ~.html
│   │   ├── forms.py
│   │   ├── models.py
│   │   └── views.py
│   ├── diary
│   │   ├── templates
│   │   │   ├── base.html
│   │   │   └── ~.html
│   │   ├── forms.py
│   │   ├── models.py
│   │   └── views.py
│   ├── images
│   │   └── [userid]
│   └── static
│       ├── css
│       │   ├── bootstarp.min.css
│       │   └── style.css
│       ├── images
│       └── js
│           └── nav.js
├── app.py
├── config.py
├── .env
└── local.sqlite
```

図 3.1 ディレクトリ構造

3.5 何ができるようになった？

3.5.1 ユーザー認証

サイトを公開する予定なので他のユーザーから自分の投稿が見られないようにする必要があり, ログイン認証をつけました (図 3.2,3.3). パスワードはハッシュ化してデータベースに格納します. 今は, パスワードの再設定が管理者しかできないので近いうちにできるようにします. 加えて Google アカウント認証も触ってみたいので実装できたらいいです.

```
@auth.route("/signup", methods=["GET", "POST"])
def signup():
    # SignUpFormをインスタンス化する
    form = SignUpForm()
```

```
5
6     if form.validate_on_submit():
7         user = User(
8             username=form.username.data,
9             email=form.email.data,
10            password=form.password.data,
11        )
12
13        # メールアドレス重複チェックをする
14        if user.is_duplicate_email():
15            flash("指定のメールアドレスは登録済みです")
16            return redirect(url_for("auth.signup"))
17
18        # ユーザー情報を登録する
19        db.session.add(user)
20        db.session.commit()
21
22        # ユーザー情報をセッションに格納する
23        login_user(user)
24
25        # GETパラメータにnextキーが存在し,値がない場合はユーザーの一覧ページへリダイレクト
    する
26        next_ = request.args.get("next")
27        if next_ is None or not next_.startswith("/"):
28            next_ = url_for("diary.index")
29        return redirect(next_)
30
31    return render_template("auth/signup.html", form=form)
32 @auth.route("/login", methods=["GET", "POST"])
33 def login():
34     form = LoginForm()
35
36     if form.validate_on_submit():
37         # メールアドレスからユーザーを取得する
38         user = User.query.filter_by(email=form.email.data).first()
39
40         # ユーザーが存在しパスワードが一致する場合はログインを許可する
41         if user is not None and user.verify_password(form.password.data):
42             login_user(user)
43             return redirect(url_for("diary.index"))
44
45         # ログイン失敗メッセージを設定する
46         flash("メールアドレスかパスワードか不正です")
47     return render_template("auth/login.html", form=form)
48 @auth.route("/logout")
49 def logout():
50     logout_user()
51     return redirect(url_for("auth.login"))
```

図 3.2 新規登録画面

図 3.3 ログイン画面

図 3.4 投稿画面

3.5.2 日記の投稿

日付, 写真ファイル, 投稿内容を図 3.4 に記入するとデータベースに userID や, 乱数で生成した写真ファイル名, 内容などを保存します. 投稿したら最新の投稿一覧に飛びます. デフォルトの日付は, 一番最後に投稿した日付の次の日になります.

```
@dt.route("/upload", methods=["GET", "POST"])
@login_required
#日記をアップロード
def upload_diary():
    form = UploadDiaryForm()
    exist_dates = get_existent_dates()
    latest_date = get_latest_date()
    date = form.date.data
    diary_text = form.diary_text.data

    if form.validate_on_submit():
        if date in exist_dates:
            flash('その日付の日記は既に存在します', 'error')
            return  redirect(url_for("diary.edit_diary",date= date))
        else:
            if form.image.data is not None:
                # 画像がアップロードされている場合の処理
                user_directory = os.path.join(current_app.config["UPLOAD_FOLDER"], str(current_user.id))
                os.makedirs(user_directory, exist_ok=True)
                file = form.image.data
```

```
                ext = Path(file.filename).suffix
                image_uuid_file_name = str(uuid.uuid4()) + ext
                image_path = os.path.join(user_directory, image_uuid_file_name)
                file.save(image_path)

                # DBに保存する
                diary = UserImage(user_id=current_user.id, image_path=image_uuid_file_name, date=date, diary_text=diary_text)
            else:
                # 画像がアップロードされていない場合の処理
                diary = UserImage(user_id=current_user.id, image_path=None, date=date, diary_text=diary_text)

            db.session.add(diary)
            db.session.commit()
            return redirect(url_for("diary.all_diary"))
    return render_template("diary/upload.html", form=form, exist_dates=exist_dates, latest_date=latest_date)
```

3.5.3 日記の編集

日記の編集は，削除ボタンが押されたときに消され，そうでない場合内容に変更になくても日付と内容は上書きします (図 3.5)．画像はアップロードすると上書きされるようにしました．改善点としてボタンを押したときに確認画面を出すようにしたいです．

```
@dt.route("/diaries/<string:date>/edit", methods=["GET", "POST"])
@login_required
def edit_diary(date):
    target_date = datetime.strptime(date, "%Y-%m-%d").date()
    form = UpdateDiaryForm()

    # GETリクエストの場合,データベースから日記データを取得
    diary = (
        db.session.query(User, UserImage)
        .join(UserImage)
        .filter(User.id == UserImage.user_id)
        .filter(User.id == current_user.id)
        .filter(target_date == UserImage.date)
        .first()
    )
    if request.method == 'POST':
        if form.delete.data:
            # 削除ボタンがクリックされた場合,日記を削除する
            db.session.delete(diary.UserImage)
            db.session.commit()
            return redirect(url_for('diary.index'))
        # フォームから新しいテキストと日付を取得
        new_diary_text = form.diary_text.data
        new_date = form.date.data

        # 日記の属性を更新
        diary.UserImage.diary_text = new_diary_text
        diary.UserImage.date = new_date
        diary.UserImage.updated_at = datetime.now()

        # アップロードされた画像ファイルを取得する
```

```
32          file = form.image.data
33          if file:
34              # ファイルのファイル名と拡張子を取得し,ファイル名をuuidに変換する
35              ext = Path(file.filename).suffix
36              image_uuid_file_name = str(uuid.uuid4()) + ext
37              # 画像を保存する
38              user_directory = os.path.join(current_app.config["UPLOAD_FOLDER"], str(
    current_user.id))
39              os.makedirs(user_directory, exist_ok=True)
40              image_path = os.path.join(user_directory, image_uuid_file_name)
41              file.save(image_path)
42              diary.UserImage.image_path = image_uuid_file_name
43          # データベースを更新
44          db.session.commit()
45
46          return redirect(url_for('diary.view_diaries', date=new_date)) # 日記一覧ページにリ
    ダイレクト
47
48
49      return render_template('diary/edit.html', diary=diary, form=form)  # 日記編集フォームを
    表示
```

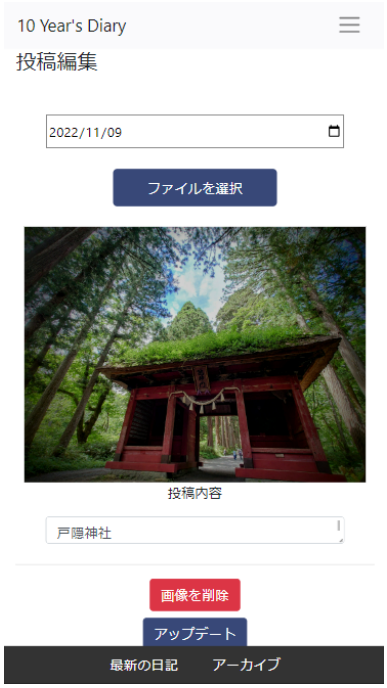


図 3.5 編集画面

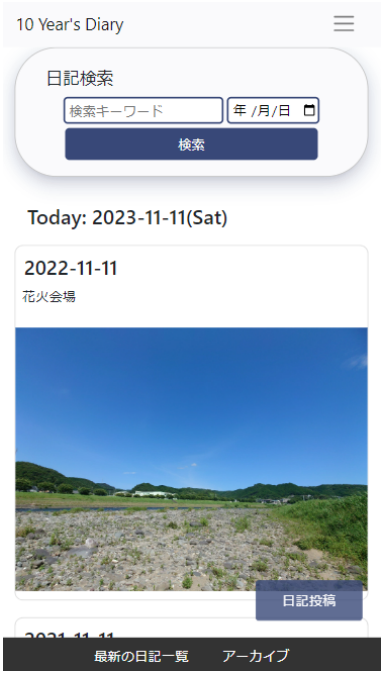


図 3.6 Top 画面

3.5.4 日記の検索

キーワード検索, 日付から過去の日記を遡って見れるようにしました. トップページに過去の同一日の投稿と検索ボックスを置きました (図 3.6).

```
# 検索ページと検索結果を表示するルート
@dt.route("/", methods=["GET", "POST"])
@login_required
def search_diary():
    form = SearchDiaryForm()
    if request.method == "POST":
        # SQLiteデータベースから日記を検索するクエリを実行
        diaries = (
            db.session.query(User, UserImage)
            .join(UserImage)
            .filter(User.id == UserImage.user_id)
            .filter(User.id == current_user.id)
            .order_by(desc(UserImage.date))
        )
        search_term = request.form.get("search_term")
        search_date_str = request.form.get("search_date")
        if not search_date_str or search_term:
            flash("どちらかを選択してください", "danger")
            return redirect(url_for("diary.search_diary"))
        if search_date_str:
            try:
                search_date = datetime.strptime(search_date_str, "%Y-%m-%d").date()
                diaries = diaries.filter(UserImage.date == search_date)
            except ValueError:
                return redirect(url_for("search_diary"))
        if search_term:
            diaries = diaries.filter(UserImage.diary_text.like(f"%{search_term}%"))
        diaries = diaries.order_by(desc(UserImage.date)).all()
        return render_template("diary/search.html", current_date=current_date, current_day=current_day, diaries=diaries, search_term=search_term,form=form,search_date=search_date)
    return render_template("diary/search.html", current_date=None, current_day=None, diaries=None, search_term=None,form=form)
```

3.5.5 一覧画面

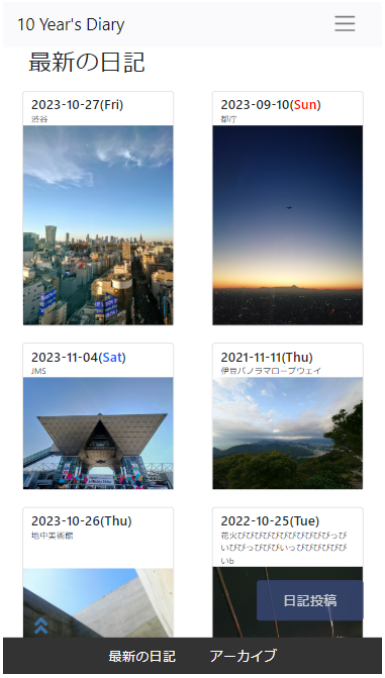


図 3.7 最新の投稿

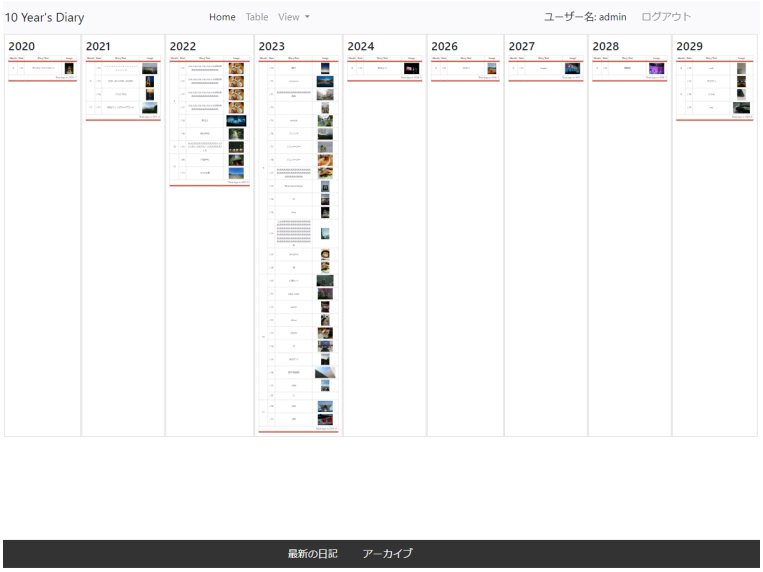


図 3.8 全投稿表示画面

最新の投稿を表示

投稿した内容を更新履歴の新しいのものから表示します (図 3.7). 投稿を選択すると編集画面に飛ぶようになっています．ここは, 重くならないようにページネーションをつけました.

全投稿表示

10 年日記を作るんだったら, 激重全日記表示ページを作りたかったので拡大すると高画素な写真を見れるページを作りました (図 3.8). 処理時間が結構掛かるのでローディング画面があると良いなと思っています.

アーカイブ

年別で過去の日記を見ることができます. 図 3.9 で選択すると図 3.10 のように月別で過去の日記を見ることができます. ここでも，投稿を選択して編集画面に移動できます.

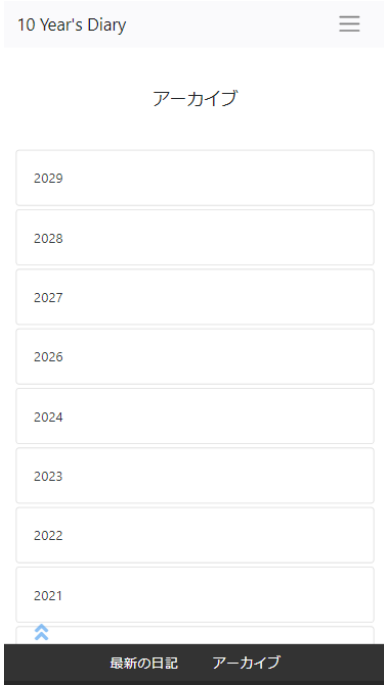


図 3.9　年選択画面

図 3.10　年別投稿表示画面

3.6　まとめ

最初は何からやっていいか分からなかったのでユーザー認証やベースの構造を Flask の公式チュートリアルを参考に進めました. 基礎ができれば全く分からないということは無く，作っていて少しづつ進化している感があり楽しかったです. 今はローカルサーバを立てて見ているので, そろそろ日常的に使えるよう家のサーバーで運用できるようにしたいと思います. 使ってみて気になるところがあれば随時更新できるのも良いです. 今後は機能追加と同じようなコードが乱立していて見にくいので, 使い回しのコードを整理していきたいです.

図 3.11　アプリのリポジトリ

第 4 章

Discord のテキスト読み上げ Bot を作ろう

koshihkari

4.1 はじめに

4.1.1 自己紹介

初めまして、今年入部した koshihkari と申します。こしひかりと呼びますが名前はこし h かりなんですね。よわよわなので色々学んでいきたいです。まだ何もそれっぽい活動はしていないということで部誌を書いてみることにしました。

4.1.2 きっかけ

僕はほぼ毎日スプラトゥーンをしているのですが、家庭の事情で通話で喋ることができません (泣)。喋れないってのは何かと不便でして、喋ってる側は discord の画面に意識を向けてないとコミュニケーションが取りにくいんですよね。もちろん僕以外の方も喋れないタイミングや理由があるわけなのでこれを何とかできないものかと考え、送信したメッセージを読み上げる Bot を作ろう！となりました。常時稼働させるのにどうしようかなと思っていましたがラズパイ買って解決しました。関係ないけどスプラトゥーンみなさんもやりましょうね、楽しいので。

4.2 やったこと

4.2.1 構想

- スプラトゥーンをするサークルで使う用として作るが、そこに所属していない友人と遊ぶ時にも使いたいので複数のサーバーで扱えるようにする
- メンションとか URL とかカスタム絵文字が送信された時は省略する
- 全員が通話から抜けた時に自動退出する機能をつける

というのを最低限の機能として作ることにしました。あとは声ですね。python の gTTS とか使ったらとてもとても簡単にできるけど、ずんだもんに読ませたいねということで非力なラズパイでも使えるように WEB 版 VOICEVOX API (高速) というサービスを使わせていただくことにしました。

4.2.2 つくる

普段は python を使ってるので今回は discord.py を使いました。
まずはサーバーの管理について。サーバーごとの情報はクラスで扱うことにしました。あとで辞書機能とか色々作るのに class じゃなくて dict で扱うと面倒そうだなっていうのとあまり使う機会がなかったので勉強がてらということで。ついでに import も書いておきます。

```
import discord
from discord import app_commands
from discord.ext import commands
import voice_download as vd

import queue


class TalkingGuild():
    def __init__(self, id: int, channnel: discord.channel):
        self.id = id
        self.reading = {channnel} # text channnel
        self.queue = queue.Queue() # message
```

voice_download.py に音声をダウンロードするコードを書いてインポートしています。ここに関しては省略します。

次に、discord.exe.commands の Cog を利用するために Cog を継承したクラスを作成します。ここで bot の機能を作っていきます。cog というのは bot の機能を複数プログラムに分割するためのものだと思ってもらえれば大丈夫です。

```
class Talk(commands.Cog):
    def __init__(self, bot: commands.Bot):
        self.bot = bot
        self.reading_guilds: list[TalkingGuild] = []

    def text_filter(self, txt: str, dictionary: dict) -> str:
        # メンションとカスタム絵文字とURLは読まない
        special_heads = ['<@', '<#', '<:'] # ユーザーメンション，チャンネルメンション，カスタム絵文字
        for head in special_heads:
            head_index = txt.find(head)
            tail_index = txt.find('>')
            if head_index == -1 or tail_index == -1:
                continue
            txt = txt[:head_index] + txt[tail_index+1:]
        if txt.find('http') != -1:
            txt = txt[:txt.find('http')]
        return txt

    def play_voice(self, voice_client: discord.voice_client, talking_guild: TalkingGuild) -> None:
```

```
        if talking_guild.queue.empty() or voice_client.is_playing():
            return
        txt, member_setting = talking_guild.queue.get()
        if txt == '':
            txt = '省略'
        vd.download(txt, member_setting, talking_guild.id)
        voice_client.play(discord.FFmpegOpusAudio(f"{talking_guild.id}.mp3"), after=lambda
    e: self.play_voice(voice_client, talking_guild))
```

音声を再生するためのメソッドと URL やメンションを無視するためのメソッドを作りました。discord 上のメンションやカスタム絵文字はメッセージを文字列として受け取ると<@625161458145034270>のようになっているので長々と読み上げられてしまいます。URL も読んでほしい人なんてまあいないので無視します。

次に bot を操作するためのスラッシュコマンドを作ります。

```
@app_commands.command(
    name = "join",
    description = "コマンド実行者が参加しているボイスチャンネルに接続し、読み上げを開始します。"
)
async def join(self, interaction: discord.Interaction):
    await interaction.response.defer()

    # コマンド実行者がボイスチャンネルに参加していないならエラーメッセージを送信
    if interaction.user.voice is None:
        title = "エラー"
        description = "読み上げを行いたいボイスチャンネルに参加してからコマンドを実行してください"
        await interaction.followup.send(embed=discord.Embed(title=title,description=
    description,color=discord.Color.red()))
        return

    # コマンドを実行したサーバーで既に使用中ならエラーメッセージを送信
    for voice_channnel in interaction.guild.voice_channels:
        if self.bot.user in voice_channnel.members:
            title = "エラー"
            description = f"このサーバーでは既に読み上げ機能が使用中です。
    \nBotが参加中のボイスチャンネル: {voice_channnel.mention}"
            await interaction.followup.send(embed=discord.Embed(title=title,description=
    description,color=discord.Color.red()))
            return

    # 読み上げ中サーバーとして追加する
    self.reading_guilds.append(TalkingGuild(interaction.guild.id, interaction.channel))
    # コマンド実行者が参加しているボイスチャンネルに参加
    voice_channnel = interaction.user.voice.channel
    await voice_channnel.connect()
    description = f"読み上げ対象チャンネル：
     {interaction.channel.mention}\n読み上げボイスチャンネル: {voice_channnel.mention}\n"
    await interaction.followup.send(embed=discord.Embed(title="読み上げを開始しま
    す", description=description, color=discord.Color.brand_green()))


@app_commands.command(
    name = "exit",
    description = "読み上げを停止し、botをボイスチャンネルから切断します。"
)
async def exit(self, interaction: discord.Interaction):
    await interaction.response.defer()
```

```
    # Botがボイスチャンネルに接続しているとき
    for voice_channnel in interaction.guild.voice_channels:
        if self.bot.user in voice_channnel.members:
            title = '読み上げを終了しました'
            description = "正常に読み上げを終了し、ボイスチャンネルから切断しました。"
            color = discord.Color.green()
            for guild in self.reading_guilds:
                if guild.id == interaction.guild.id:
                    self.reading_guilds.remove(guild)
                    del guild
                    break
            else:
                description = "読み上げ中サーバーのリストからの削除に失敗しました。"
                color = discord.Color.red()
            await interaction.guild.voice_client.disconnect()
            await interaction.followup.send(embed=discord.Embed(title=title,description=
    description,color=color))

    else:
        await interaction.followup.send('
    Botはこのサーバーのボイスチャンネルに参加していません。')
```

join コマンドを使うとコマンドを実行したユーザーが参加しているボイスチャンネルに接続します。exit コマンドを使うとボイスチャンネルから bot を切断できます。ここに関しては特に工夫したこととかはないです。

次にメッセージを受け取り、再生までの機能を実装します。commands.Cog.listener でデコレートすることで discord 上のイベントを扱えるようになります。この bot ではメッセージを受け取った時に発生する on_message、ボイスチャンネルの状態が更新された時に発生する on_voice_state_update を使っていますがリアクションをされた時のイベントなど他にも色々あるので気になったら調べてみてください。

```
@commands.Cog.listener("on_message")
async def read(self, message: discord.Message):
    if message.author.bot: # メッセージ送信者がbotの場合
        return
    # Botがボイスチャンネルに接続していなければメッセージを無視
    for voice_channnel in message.guild.voice_channels:
        if self.bot.user in voice_channnel.members:
            break
    else:
        return
    guild_id = message.guild.id
    text_channel = message.channel
    # サーバーの取得
    for talking_guild in self.reading_guilds:
        if talking_guild.id == guild_id:
            # 読み上げ対象外のチャンネルの場合無視
            if not text_channel in talking_guild.reading:
                return
            break
    txt = self.text_filter(message.content)
    talking_guild.queue.put(txt)
    self.play_voice(message.guild.voice_client, talking_guild)
```

discord.Guild.voice_client の play メソッドではボイスチャンネルで音声を再生できますが、これだけでは音声再生中に受信したメッセージは再生することができません。それを解決するために after に再生用の関数を与えます。after に与えられた関数は再生後にさらに実行されるため、メッセージのキューが空になるまで再生が続きます。

最後に自動退出機能を作ります。

```
@commands.Cog.listener("on_voice_state_update")
async def auto_hung_up(self, member: discord.Member, before: discord.VoiceState, after:
    discord.VoiceState):
    if member.guild.voice_client is not None and len(member.guild.voice_client.channel.
    members) == 1 and member.guild.voice_client.channel is before.channel:
        await member.guild.voice_client.disconnect()
        for guild in self.reading_guilds:
                if guild.id == member.guild.id:
                    self.reading_guilds.remove(guild)
                    del guild
                    break
```

これによって exit コマンドを使い忘れても bot が通話に残り続けることがなくなりました！

4.3 使ってみて

めちゃくちゃ快適になりました。ちょっと声がデカくて他の人の会話を遮る性能が高いですが、まあそこのところは各自 discord で設定できるので問題なしということで、、。僕以外の方にも使ってもらっていて嬉しい限りです。

ここに書いたもの以外にも辞書機能、声のスピードを変えたり声を変えたり、色々機能を追加したのですが使いやすいシンプルなものを目指すとここに書いた機能だけで十分でした。

4.4 最後に

日本語下手な僕ですがここまで読んでいただきありがとうございます。作成したものの画像を最後に置いておきます。せっかく買ったラズパイ、もっと使いたおしてやりたいと思います！

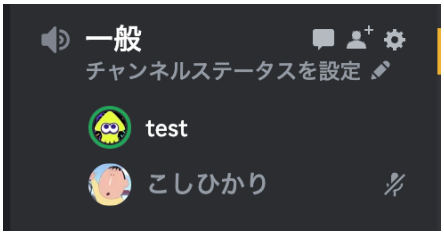


図 4.1 ボイスチャンネルで再生中の bot

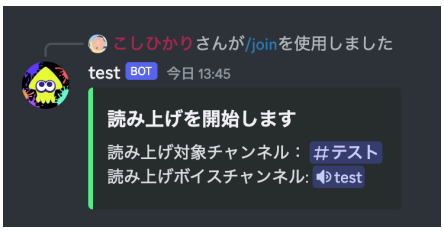


図 4.2 join コマンド実行時のメッセージ

第 5 章

プログラミング言語初心者がプログラミング言語を作ってる

ryota2357

こんにちは。2 年の ryota2357 です。

最近 Rust でプログラミング言語を作ってます。

今回の部誌で、その作ってるプログラミング言語について書きたかったのですが、あまりにも開発が遅れていて「こんな言語作ってるよー」って言えるものがありません。そこで、「こんなこと頑張ってるよー」ってことを書きます。

5.1 はじめに

タイトルに書いた通り僕は「プログラミング言語初心者」です。

「プログラミング言語初心者　#とは」って感じですが、意味は「何かの言語の仕様を詳しく知っている」や「色んな言語の特徴を知っていて書いたことがある」とか、そうゆうプログラミング言語への理解が少ない、と言う意味で僕が勝手につけました。

僕が書いたことある言語（1 ヶ月以上の何かの開発・学習等で利用した）は「C, C++, C#, Dart, JavaScript, Lua, Python, Rust, Swift, TypeScript, Vim script」で、その中でも最低限は使えると自分で思ってるのは C#だけで、これも Unity で長く使ってたからと言うかなり限定された理由です。(Swift 至っては本当に一瞬書いてただけってレベルだし...)

さて、この言語の並びから気がついた方もいるかと思いますが、僕は関数型言語に触れたことがありません。ちょっとこれは僕自身もどうなのかな？と思っていて、ちょくちょく F#を見たりしてはいるのですが、全く馴染めてません。なので、現在開発している言語も関数型言語ではなく、手続き型言語です。

あと僕は型とか良くわからない（TypeScript とか雰囲気で書いてる）ので、型推論を最悪しなくても良い動的型付け言語を作ってます。

さて、そろそろ本題に入りましょうか。ここからは、この僕レベルの初心者がプログラミング言語を作るとどうなるのか、何をしてるのか、を書いていこうと思います。

5.2 作り始める前に

自作のプログラミング言語で書かれたスクリプト (テキストファイル) はどうやったら動くのか、そもそも僕はここからあんまり良く分かってませんでした。

(僕) なんか Token にしてー、良くわからん木構造にいい感じにしてー（AST って言うんでしょ、聞いたことあるよ！)、その木をいい感じにコンパイルなりしてー...

これに関して僕は非常に運がよかったです。

僕はたまたまちょうど、「正規表現エンジンって DFA っていうオートマトンで作れてね...」って話を 1 年の後期くらいで教えてくれた先輩がいました (mado さんありがとう)。それで、僕は正規表現エンジン作ってみたいなーと思ってました。そして今年の 5 月に作りました。

これがまず、初期段階においていい経験となりました。

この時作った DFA 型正規表現エンジンの実装では、Lexer と Parser も書きました。(Lexer はエスケープ処理のみ、Parser は再帰降下法と言う非常にシンプルなもの) これらを書いて、あれ、「意外と構文木作るのって簡単なのでは？」ってなりプログラミング言語を作る上で直近にあった難しさをだいぶ感じなくなってました。

あと、時期は忘れてしまったけど、mado さん (さっきも出てきたね) に chumsky って言うパーサーコンビネーターを紹介してもらって、意外とできそう！？なんて思ったりもしてた。

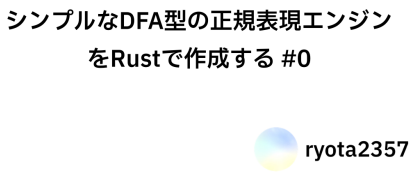


図 5.1 記事 OG 画像

図 5.2 https://ryota2357.com/blog/2023/dfa-regex-with-rust-0/

ちなみに、この正規表現エンジンは僕が初めて Rust で自分で実装したプロジェクトとなりました。

5.3 開発

開発記録を雑に書きます。

5.3.1 〜3 ヶ月前

何はともあれ、プログラミング言語作ってみようと適当に chumsky（Rust 製のパーサーコンビネーター）をいじり回し始めました。

もう、ここには書ききれない苦労がありました。

そもそも Rust を僕は良く分かってなくて、「trait？、C#の interface みたいなものか...」「スライス？、C#の Span<T>みたいなやつね」「所有権？、C++ で move とかあったね」みたいに雑な思考で書いていたのでずっとコンパイラに怒られまくってました。

Lexer、Parser、構文木の直接実行機（これは結局使いませんでした）、VM（Compiler で独自の VM コード列吐いてそれを読む）、を色んな方法で作っては、「この実装じゃだめだ。1 からやり直し！」を繰り返してました。（どれも 5 回以上...）

ちなみに少し後のことなのですが、Compiler は作り直し 0 回でできました。これは VM の実装のために人間 Compiler やってたからです...

そろそろ作り方決まったかな？と言うタイミングで GitHub にリポジトリを作りました。

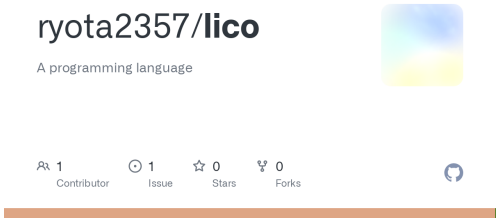


図 5.3 リポジトリ OG 画像

図 5.4 https://github.com/ryota2357/lico

現状名前は「Lico」と名付けてます。名前の由来は書きませんが、3〜4 文字で 2 音と言う制約の元で考えました。

5.3.2 〜1 ヶ月前

作り方決まったはずだったのにちょっと難航して、1 ヶ月前くらいにやっと、最低限動く何かが見え始めました。

僕は最低限動くものができたらそれを v0.1 として「できた！」って言おうと決めています。（そうじゃないと永遠に完成しないので）

そこで v0.1 までの TODO と v0.1 まではやらないこと書き出しました。

Loadmap to v0.1 #1

Edit New issue

Open 4 of 11 tasks ryota2357 opened this issue 3 weeks ago · 0 comments

ryota2357 commented 3 weeks ago • edited Owner

Todo

- [ ] Create CLI to compile from the program source code file and run it.

Lexer

- [x] String escape sequence. (`6703fb1`)
- [ ] Line comment.

Parser

- [ ] Minimal parse error handling. It uses a rough recovery strategy.
- [x] Add more Span to Stat/Expr AST. This is the preparation for LS, Linter, etc that uses the span for searching/walking on the AST. (`7d8d8c5`, `a50f057`)

Compiler

- [x] Compile basic AST to VM code without extra. For v0.1, the Compiler cannot compile a program containing attribute statements. (`85248ac`)
- [ ] Error handling for invalid operations such as break/continue outside the loop.
- [ ] VM code testing. (Lexer-Parser-Compiler-VM integration test?)

VM

- [x] Minimal parse error handling. VM returns `Result<Object, String>`. The `String (Err)` is not organized and is not useful for generating error messages. (`9e17a42`)
- [ ] Add basic operation instructions. For v0.1, some meta method (such as `__add()`) does not work.

Document

- [ ] Deploy empty content.

After v0.1

- Separate some functions into a `core` crate.

Lexer

- Multi-line string.

Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone: No milestone
Development: Create a branch for this issue or link a pull request.
Notifications: Customize
Unsubscribe
You're receiving notifications because you're watching this repository.
1 participant
Lock conversation
Unpin issue
Transfer issue
Delete issue

図 5.5　11/19 時点の ryota2357/lico#1

5.3.3　〜2 週間前

FizzBuzz が動きました！

```
var limit = 20
var i = 0
while i < limit do
    i = i + 1
    if i % 3 == 0 and i % 5 == 0 then
        print("FizzBuzz")
    elif i % 3 == 0 then
        print("Fizz")
    elif i % 5 == 0 then
        print("Buzz")
    else
        print(i)
    end
end
```

```
Running
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
```

図 5.6　自作言語で FizzBuzz

もう、なんかめっちゃ嬉しかったです。こんな素人でも動くプログラミング言語が作れるのか！ってなりました。

この FizzBuzz ですが見ての通り while 文で実装されてます。今まさに for 文でかけるように VM 側に実装を色々足してます。

あと、少しだけこの言語の仕様に触れると、この言語は Lua に大きく影響された言語です。関数などのインターフェースは C#や Ruby に影響されたものとなる予定です。内部実装（VM の実装）は高レベルな機能を持ったスタックマシーンベースの何か、と言う感じです。

これ以降の開発の記録は僕の「しずかなインターネット」に流していく予定なので、気になった人は Lico のリポジトリと「しずかなインターネット」見てもらえたらな、と思います。

5.4 感想？

いやー、Rust っていい言語だな、と言うのをプログラミング言語作って、しみじみ感じています。

Rust で作ると設計をだいぶ考えさせられます。これは僕がオブジェクト指向な方針を取りやすいと言うのもあって本当に大変でした。クラスやインターフェースを用いた抽象化は直感的で雑に作れるのですが（もちろん綺麗な設計は難しいけど、規模が小さいうちは少し雑でも良さげな感じになる）、Rust でそれをやると Clone なり dyn なりが大量に出てきたり、そもそも実現不能（ライフタイム制約）だったりして色々な実装方法を試しては失敗してを繰り返しました。

まだまだこのプログラミング言語 Lico の開発は初期段階です。これからも開発は長く続きます。また何かの機会で僕の言語について紹介できたらいいなと思っています。

最後に、ここまで読んでくださりありがとうございました。

第 6 章

カナダ旅行 (留学) とウェブサイト作成の話

furatto

6.1 はじめに

こんにちはフラットです. 2 週間前に入部したての身ですが早速部報を書きました. 入部したてですがモノづくりサークル自体は 1 年のころから入っており, 工研の方で細々と活動していました. 2 年からソフトウェアの方でもっと活動したいと思い立ち,MMA に入った所存です. これから毎年部報を上げていくつもりですが, もしいなかったらあいつサボってんなと思ってください. 多分サボってます.

6.2 カナダ旅行 (留学) 行ってきた

3 週間だけの短期間ですが夏にホームステイで楽しんで参りました. 思ったより円安が酷くて次の月の請求がとんでもないことになったのもいい思い出?になりました. 写真を交えつつ (メインで) 向こうの話をしようと思います.

6.2.1 1 週目

まだお金の余裕がある週です.2 週目で破産,3 週目でとどめを刺します. 図 6.1 が初日に取った写真, ホームステイ先の家の前ですね. 家も道路もバスも何もかも, とにかくすべてがデカかった. カナダのバスは中央で連結していて電車みたいに曲がるようになってるタイプでした. 初めて見た. 電車は逆に日本と比べるとかなり質素でした. まず車掌がいない. 完全自動運転で椅子も少なめで, サイズは同じでしたがかなり広く感じました. あんまり長い編成の電車はいませんでしたね. エリザベス女王の公園にも行ったのですが,Park って書いてたのに着いたら mountain が鎮座しててたまげました. そして図 6.2 が私が留学した先の大学,UBC です. バカみたいにデカい. デカすぎてビーチが 2 つ大学内にありました. あほ. 他にも博物館やらが 2,3 棟. 図書館が 4,5 棟あり, 森の中ではスカイウォークもできたみたいです. とんでもなく設備やらが豪華なのですが代わりに学費もとんでもないらしく,UBC の学費は家と変わらんとステイ先の人がぼやいてました. 図 6.3, ラーメン屋もありました. おいしかったです.

図 6.1 家前

図 6.2 UBC の海

図 6.3 ラーメン

6.2.2 2 週目

破産する週です. 覚悟はいいか? 円安がやばいと言いましたが, やばい中私は外食を楽しんでいました. Tim Horton っていうカナダの有名なドーナツ屋が安いわりにおいしくて結構食べたり, 良さげな個人経営のバーガー屋を見つけて食べたりととにかく食べてました. ステイ先から朝昼晩の食事は提供されるのですが, カナダで食べまくった結果胃が肥大化してしまい, 一日 5 食ぐらい食べてました. 破産の原因です. でも日本戻ったら胃は縮小しました. 不思議ですね. この週も食べることを欠かさず, ドーナツ食ったり図 6.4 のジャパニーズ・Bento(カリフォルニアロール付き. おいしかった) を食べたりと満喫している中. 私はバンクーバーの首都. ビクトリアへ行くことを思い立ち, フェリーで行ってきました. バスで 2 時間. フェリーで 1 時間ほど揺られて到着. 結構古い街並みが残る場所で, おいしいと勧められたスパゲティ屋行ったり, 有名な建物みたりと観光を満喫しました. この日は日が変わってから帰宅. 治安良くて助かった. そんな金使ってねぇじゃんって思われるかもですが, 全然服買ってたし色々食いまくってたので問題ありません (?).

図 6.4 Bento

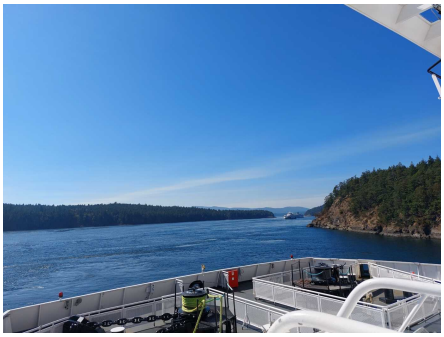


図 6.5 フェリーなう

図 6.6 スパゲティ

図 6.7 なんかの有名な建物

6.2.3 3 週目

とどめを刺しました. 後悔はしてません. とはいえこの週はそんなに出費した日は全然ありました. お金も少なくなってきたので外食も普通に行ってました. ばか. ところで外国といえば何を思い浮かべるでしょうか? そう!銃ですね! 知り合い誘って撃ってきました. 図 6.8 図 6.9 をご覧ください. 実銃ですよ実銃. 流石にフルオートは許されませんでしたがめちゃ楽しかったです. せっかくということで一番威力が高いリボルバーとショットガンの弾を注文した所、, 体幹が弱すぎて吹っ飛びました. 知り合いに見せたら弱そうだのモブ A だのボロカス言われました. 許さない. 個人的に驚きだったのはショットガンよりリボルバーの方が衝撃が強かった事ですかね, 手に衝撃が集中でもしたのか 3 発目でめちゃ手痛くなりました. その後は知り合いとハンバーガ食って解散 (図 6.10) おかわりもした.

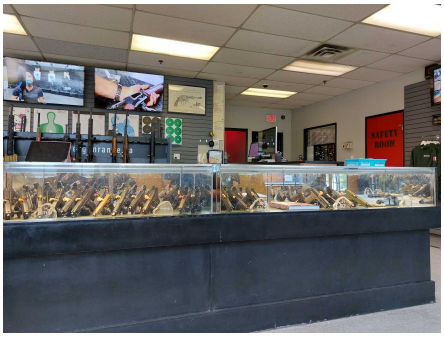


図 6.8 銃!

図 6.9 リアルショットガン

図 6.10 ハンバーガー

そんなこんなでカナダ旅行 (留学) 終了!後で見たら毎日 10km ぐらい歩き回ってたらしいです. 十分満喫したと言えるでしょう. 出先で会った方々ともカジュアルに話せたし, 個人的には大満足です. 一つ心残りがあったとすれば,3 週間が思っていた以上に短かった事ですね. 次は長期のに挑戦したい. ほんとは Web の話もしたかったんですけど長くなりそう (完成もしてない) ので今回はここまで. 多分文化祭までには完成すると思われ. あなたもカナダ旅行どうでしょうか? めっちゃリスいるのでリス好きにもたまらない空間ですよ. なんでも書いていいって言われたからこれ書いたんすけどダイジョブかな? あとで見た目みて変だったら次回の部報でもっときれいにしようかな.

第 7 章

HowToHandaduke

daiki

7.1 はじめに

こんにちは。1 年生の kanakana(または daiki) です。普段はロボットをやっているので、触っている内容はどちらかというとハードウェアに比重が偏ってますが、競プロもほんとに少しだけやってます。最近発注した基板が大陸から大量に届いて、はんだ付けをよくするのではんだ付けについて書きます。はんだ付けしたい人、もしくははんだ付けされたい人は参考にしてみてください。

7.2 挿入実装部品のはんだ付け

穴に部品の足を入れてはんだ付けするタイプの部品を挿入実装部品といいます。挿入実装部品のはんだ付けは、以下の順序で行います。

1. フラックスをランド (部品を挿す穴の周りの導体部部) に塗る。
2. ランドに小手先を当て、ランドを温める
3. ランドに小手先を当てたままはんだを当て、溶かす。この際、はんだが芋はんだにならないようにする。

芋はんだの基準は感覚です。多分調べると図が出てきます。フラックスとは、いわゆるヤニ[*1][*2]のことで、金属表面の酸化被膜を取り除くことではんだ付けをしやすくしてくれる薬品です。フラックスを使うか使わないかで、はんだの広がりやすさはかなり変わります。また、表面張力を弱めてくれるので、はんだブリッジすることが少なくなります。(フラックスに含まれるロジンは気化したものを吸いすぎると健康に害を為す可能性があるので、はんだ付けの際は窓を空けます。)

[*1] 天然物のヤニとして松ヤニがあるが、戦時中に燃料として活用しようとしていたやつとは別物らしい。
[*2] 松ヤニは重量挙げの人が持ち上げる前にパンパンしてる粉に使われてるらしい。

7.3 表面実装部品のはんだ付け (はんだごてを用いる場合)

表面実装部品とは、挿入実装部品とは異なり、穴を通すことなく基板の片方の面だけではんだ付けが完結する部品のことです。マザーボードなど製品の基板でよく見ます。

7.3.1 SOP,SOJ,QFP,QFJ パッケージなどのはんだ付け

表面実装部品の中にも様々なパッケージの規格があり、その中でも部品の足が上から見た時に見えるもので代表的なのが SOP,SOJ,QFP,QFJ パッケージです。SOP,SOJ は大体図 7.1 のようなタイプで、QFP,QFJ は図 7.2 のようなタイプです。

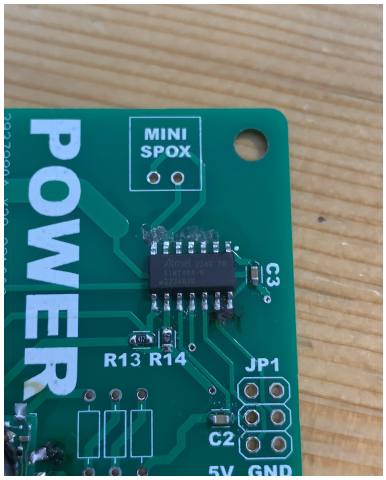


図 7.1 SOP の例

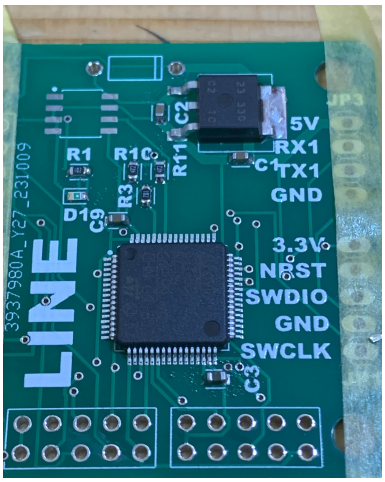


図 7.2 QFP の例

これらのはんだ付けの手順は以下のようになります。

1. 部品をマステもしくは仮はんだ (1〜3 ピンだけはんだの量少なめではんだ付けすること) で固定する。個人的にはマステの方がやりやすい。
2. フラックスを足らへんに塗る。
3. ランドの外側の部分から滑らすようにコテを移動させて、コテ先を端子の先に当てた状態ではんだを流す (同時に 3 ピンずつぐらいやる)。
4. はんだが足の下の方まで行き渡るのがなんとなく分かる、分かったら隣にずれる。

これらのはんだ付けをする際は、コテ先が平べったくなっているものを使うとやりやすいです。

また、このような端子間のピッチが非常に細かい部品をはんだ付けするときは、出来栄えの確認にルーペが便利です。[*3]

[*3] ガチ勢は立体顕微鏡を使う。

図 7.3　父親から拝借したルーペ

7.3.2　チップ抵抗、コンデンサ系のはんだ付け

チップ抵抗、コンデンサは、その名の通りチップ型の抵抗、コンデンサです。図 7.4 にチップ抵抗、チップコンデンサが映っています。

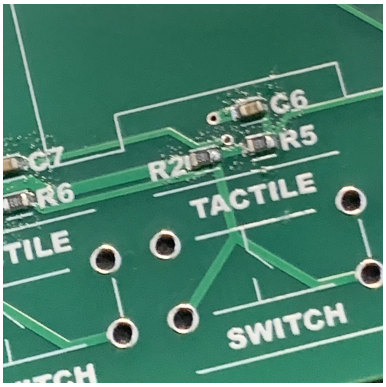


図 7.4　チップ抵抗、チップコンデンサの例

これらのはんだ付けの手順は以下の通りです。

1. 2 つのパッドのうちの片方にはんだを流す。
2. はんだを流したパッドを温めながら、部品をピンセットで横からずらして位置を合わせ、はんだ付けする。
3. もう片方をはんだ付けした後、最初の方にはんだを継ぎ足す。

2 で横からずらすときに基板から浮かないようにすると、はんだ付けが終わったときに浮かなくて済みます。浮いたままはんだ付けしてしまうと、発振などの原因になるらしいです。

7.4 表面実装部品のはんだ付け (ホットプレートを用いる場合)

実際に工場で部品を実装する際は、はんだと部品を載せた後に基板ごと温めてはんだ付けするリフローという手法が用いられます。工場にあるリフロー炉をホットプレートで代用することで、一般 (諸説あり) の家庭でも行うことができます。手順は以下の通りです。

1. クリームはんだをパッドに塗る。普通はステンシルを用いるが、お金がない場合つまようじでもいける。
2. 部品を正確に載せる。
3. ホットプレート上で加熱する。180 ℃で体感 20 秒くらい待つ。

図 7.5 ホットプレートで加熱している様子

図 7.6 クリームはんだの例

クリームはんだとは、図 7.6 のような注射器型の容器に入っているクリーム状のはんだのことで、これを塗った上に部品を載せて熱した後に冷やすとはんだ付けができます。この容器をラベルなしで持っていると何も知らない人はヤクと見分けがつかないので、ラベルははがさないようにします。[*4]

また、ステンシルというのは、部品のパッドの形に穴をあけたアルミの板のことで、これを基板のパッドと合わせた後、ステンシルの上からクリームはんだを大まかに塗ると、きれいにパッドの上だけにはんだを塗ることができるというものです。これがなくても、つまようじにクリームはんだをとって、気合で塗ることもできます (しんどい)。私はつまようじでやりました。

[*4] これのせいで家族会議になった人を前ツイッターで見た。

7.5 終わりに

誰得なのか全くわからないことを書きましたが、はんだ付けのやり方が分かってると基板を作る心理ハードルが低くなります。ソフトウェアだけでなく基板も作れると IOT 的なこともしやすくなると思うので、知っておいて損はないかもしれないです。僕はソフトウェアの方がまだまだですが、、、サーバーとかも触ってみたいです。

ありがとうございました。

百萬石 2023 -秋- © 電気通信大学 MMA

2023 年 11 月 24 日　初版第一刷発行　【本書の無断転載を禁ず】

著　者　gae, Udon, namor, koshihkari, ryota2357, furatto, daiki
表　紙　Udon
編集者　Udon
発行者　電気通信大学 MMA
発行所　電気通信大学 MMA 部室
〒182−8585 東京都調布市調布ヶ丘 1−5−1 サークル会館 208 号室
https://www.mma.club.uec.ac.jp/
印刷所　電気通信大学 MMA 部室
製本所　電気通信大学 MMA 部室

電気通信大学