

WORD

From College of Information Science



WORD編集部は除草剤を
撒いていません号

目次

SATySFi で BibTeX の parser を書く	金子尚樹 (<i>puripuri2100</i>)	1
バカとバスこそインターんに行け！	<i>maetin</i>	21
大学内で世界一臭い缶詰を合法的に開けた話.....	<i>cely chan</i>	25
メールバトル 2023	<i>raspi0124 & momeemt</i>	31
Openfab 創房を知っておこう	てばさき (@tebaskit)	41
Linux で絵を描いてみよう	高尾創介 (<i>omemoji</i>)	48
原動機付自転車はいいぞ	<i>onokatio</i>	52
オタク京都旅行記	<i>yuseiito</i>	55
宅配物の状況を CLI で一元管理する	<i>lapla</i>	61

SATySFi で BibTeX の parser を書く

文 編集部 金子尚樹 (puripuri2100)

1 はじめに

前回の「WORD 編集部は 44 年目も営業しています号 (53 号)」で筆者は「BibTeX フォーマットをパースしたい」というタイトルで記事を書いた。前回は BibTeX の構文を紐解き、OCaml で parser を作るための方法について示唆するにとどましたが、今回は SATySFi^{*1} という組版用プログラミング言語を用いて BibTeX の parser を書く具体的なコードについて解説をする。

2 BibTeX とその文法について（前回の復習）

BibTeX とそのフォーマットについては前回号で詳しく書いたため、ここでは軽く触る程度とする。

BibTeX は参考文献管理用のフォーマットの一つであり、主に LaTeX で使われることを想定している。理工系の分野では多く使われており、例えば arXiv では論文情報の自動生成は BibTeX 形式のみをサポートしている。形式としては

```
1 @article{id1,
2   tag1 = "text",
3   tag2 = {text},
4   page = 2,
5 }
6 @comment {
7   comment 1
8   comment 2
9   comment 3
10 }
11 @misc{id2,
12   tag1 = "text",
13   tag2 = {text},
14 }
```

のようなものが一般的で、

^{*1}<https://github.com/gfngfn/SATySFi>

- @のあとに「資料の発表形態がなにか」を表す文字列を入れる（エントリと呼ばれる）。以下のようなものが最初から用意されて広く使われているが、規則上は空白を挟まない文字列であればよい。また、アルファベットの大文字と小文字は区別しない。
 - article : 雑誌
 - book : 書籍
 - phdthesis : 博士論文
- エントリの後の波括弧で括られたブロックの中にはカンマ限りの Key-Value リストが書かれる。ただし、開き波括弧の直後には参考文献に振る一意の ID を文字列で指定する。Key-Value リストの中身は以下のようなものが広く使われているが、エントリによって必須のものや任意のものなどがある。また、これもエントリと同じように規則上は任意の文字列でよく、大文字と小文字の区別はしない。
 - author : 著者名
 - title : タイトル
 - year : 出版年

といったフォーマットになっている。

ただし、BibTeX 形式では Key-Value リスト形式の Value 部分にあたる“テキスト”的構文がややこしいため、注意が必要である。具体的には下記の項目のような難しい点が存在している。

- テキストの範囲はダブルクオーテーションまたは波括弧
- テキスト内で波括弧で括られた箇所は加工禁止となり、分割などができる
- `#`記号でテキスト同士の結合ができる
- テキストの定義ができる
- LaTeX コマンドを使うことができる

より詳細な文法については BibTeX 公式サイト^{*2}が詳しい。

3 SATySFi について

SATySFi とは新しい組版用プログラム言語および組版処理ソフトウェアのこと、LaTeX 風の構文で記述することができるマークアップ層と OCaml 風の構文を持つプログラム層を持ち、静的な型付けと型検査を含む静的な検査機構をもっていることが特徴である。

例えばマークアップ層は

```
1 +section {Section} <
2   +p{段落1\href(`https://github.com/gfngfn/SATySFi`){SATySFi の
```

^{*2}<http://www.bibtex.org/Format/>

```

1     リンク}]}
2     +enumerate {
3         * 箇条書き1
4             ** 箇条書き1.1
5         * 箇条書き2
6             ** 箇条書き2.1
7                 ** 箇条書き2.2
8
9     }
10    +p{数式も書ける:$\{\sqrt{a^2 + b^2} = c\}}
11    +subsection {Sub section} <
12        +p{段落}
13    >
14 >
```

のような記述をする。

- 縦方向に組まれる要素についてはプラス記号から始まるコマンドを使ったり角括弧で括ったりする
- 横方向に組まれる要素についてはバックスラッシュ記号から始まるコマンドを使ったり波括弧で括ったりする
- 数式はドルマークから始まる波括弧で括る

という原則を理解するとそこまで複雑な構文になっていないことがわかる。

プログラム層はこのマークアップ層で記述されたテキストを加工し、PDF に配置するためのオブジェクトを生成する層である。

たとえばテキストのリストを縦方向に重ねていくコマンドは

```

1 let-inline ctx \stack lst =
2   lst
3   |> List.map (fun text -> read-inline ctx text)
4   |> line-stack-top
```

のよにして定義することができ、マークアップ層ではこのコマンドを

```

1 +p{
2   \stack[
3     {The};
4     {quick};
```

```
5 {brown};  
6 {fox.}  
7 ];  
8 }
```

のようにして使うことができる。実行結果は図 1 のようになり、意図しているようにテキストが上下に積み重なった状態になっていることがわかる。ここではコマンドの定義に出てきた `line-stack-top` という関数が肝となっている。この関数の型は `inline-boxes list -> inline-boxes` であり、「横方向に向かって配置されるオブジェクトのリストの中身を縦方向に積み上げ、一つの横方向に向かって配置されるオブジェクトにまとめる」という機能を持つ。実際のコマンドの定義ではこのほかにもスペースや文字サイズを変更するような関数を用いて様々な調節を行うことで、例えば

```
1 今日も\ruby ?: [Ruby.mode Ruby.g] [`しぐれ`] {|時| 雨|} が降る。
```

と書くと図 2 のように表示されるようなルビを実現するためのコマンドを定義することができる。このコマンドは 700 行程度で実装しており、SATySFi-ruby リポジトリ^{*3} で公開している。

The text "The quick brown fox." is displayed in a large, bold, serif font. The words are stacked vertically: "The" on top, followed by "quick", "brown", and "fox." at the bottom. The letters are slightly slanted to the right.

図 1: stack コマンドの実行結果

*3 <https://github.com/puripuri2100/SATySFi-ruby>

今日も時雨が降る。 しぐれ

図 2: ルビを振るコマンドの実行結果

このように SATySFi はプログラム層で組版部分に介入をしつつ、マークアップ層ではプログラム層で定義された関数を呼び出しながら人が意味だけに集中して文章を書いていくことができるようになっている。また、SATySFi の構文は BNF で記述できるような構文規則であるため、解析が簡単である。これに対して LaTeX はコマンドの展開結果によってはどの文字が特殊な役割を持つのかさえも変動するため解析が非常に困難となっており、フォーマッタやコンバータの作成が不可能である。このように、記述された内容を静的に解析ができるることは大きな利点となっており、LaTeX よりもエラー報告がしやすく、人も機械も書きやすいという特徴に繋がっている。

4 SATySFi で BibTeX の parser を書く利点

SATySFi は「最近の知見を取り入れたより良い LaTeX」として開発された側面がとても強く、よく言われる「わかりやすいエラーメッセージ」という利点や前述した静的な構文解析が可能な文法のほかにも、モジュールシステムや静的型、マクロ機能などの LaTeX ができたあとに発展を続けたプログラミング言語作成についての知見や Unicode を前提とした多言語組版や JLReq などの組版処理についての知見を活用して設計されている。そのため利用者の層はかぶっており、既存の LaTeX ユーザーに SATySFi を利用しようと思ってもらうためには、少なくとも表面的にでも LaTeX で使える便利な機能は当然 SATySFi でも使える状態であることが望ましい。

既存の LaTeX の利用者が LaTeX の何を便利だと思っているかについては様々な意見があるが、インターネットや本で主張されている内容や筆者自身の経験を勘案すると以下の項目におおよそ収束すると考えられる。

- テキストだけで文書を作成することができる
- 文書の体裁を制御する設定ファイルを利用して執筆者は本質的な文章のみに集中することができる
- 文献管理と自動参照が楽にできる
- PDF の制御ができる

このうち、SATySFi では「テキストだけで文書を作成することができる」と「文書の体裁を制御する設定ファイルを利用して執筆者は本質的な文章のみに集中することができる」については実現しており、文献管理と PDF の制御で LaTeX と同等の利便性を提供することが重要である。PDF の制御についてはバックエンドの PDF 生成ライブラリその

ものに手を付ける必要がありハードルが高いため、ひとまず文献管理の機能向上を図ることにした。

SATySFi で文献管理と参照を行う場合、文書の体裁を管理する「クラスファイル」と呼ばれる設定ライブラリが提供する機能を使う他に BiByFi^{*4} というライブラリを用いる方法がある。ただし、いずれの場合にでも文献情報を SATySFi のデータ構造として与える必要があり、すでに膨大な文献リストが存在する人にとってはその乗り換えコストはとても小さくない。筆者は BibSATySFi^{*5} という「BibTeX ファイルを BiByFi が期待するデータ構造を表すファイルに変換するソフトウェア」を作成したが、インストールの手間があるほか、文献管理ライブラリが期待するデータ構造に破壊的な変更が入った場合にはもう一度変換し直す必要があり、利用しにくい選択肢となっていた。

そこで、SATySFi が提供するプログラミング言語としての機能を使い BibTeX ファイルを解析し、ライブラリ側で求めるデータ構造への変換を行うようにすれば、少なくとも利用者側からは既存の BibTeX ファイルを与えるだけで使える状況を作り出すことができると考え、実装することにした。

5 実装の方針

前号でも述べた通り、BibTeX フォーマットには

- 波括弧に以下の 4 つの意味がある
 - フィールドの開始
 - テキストの開始
 - テキスト内の「加工禁止」の開始
 - テキスト内のコマンドの引数の開始
- アクセント記号を表すコマンドの処理が必要

という特徴があり、特に一つの記号が複数の意味を持つのは構文解析を行う上で考慮すべきことが増え実装が難しくなる要因となる。そこで、構文解析時に区別がしやすいように一つの文字を文脈に応じて違うトークンに変換したり複数の文字をまとめて一つのトークンに変換したりする字句解析器^{*6}を作り、これを間にかませることで構文解析が単純な再帰関数で実装できるようとする。

構文解析器では字句解析器が生成したトークンの列を先頭から読んでいき、トークンを消費してエントリを生成することを繰り返し、終端に達したら処理を終了するような実装となる。

入力としての文字列を字句解析器と構文解析器で処理し、最終的にデータ構造を生成する流れは図 3 のようになる。

*4 <https://github.com/namachan10777/BiByFi>

*5 <https://github.com/puripuri2100/BibSATySFi>

*6 lexer ともいう

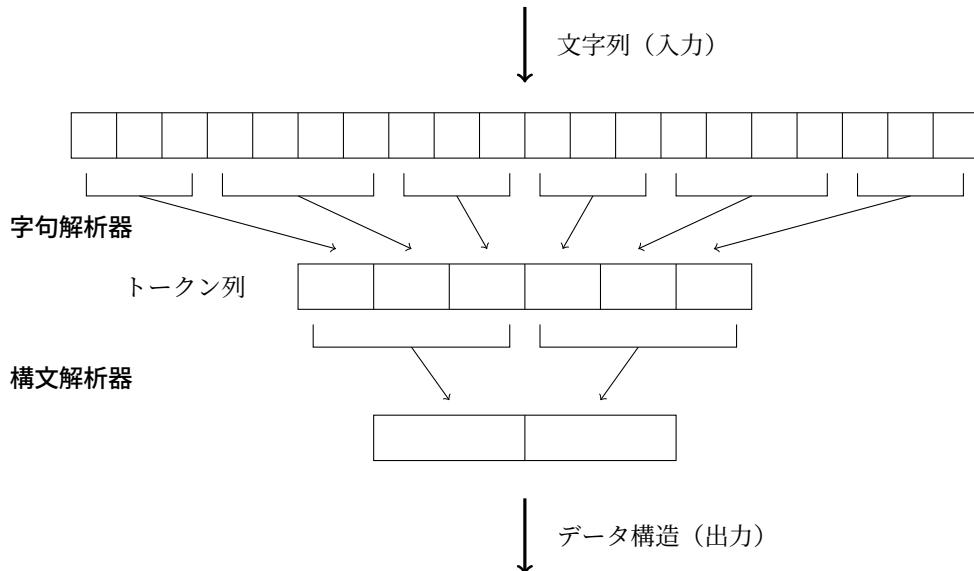


図 3: 文字列を受け取り字句解析器と構文解析器で順に処理をしてデータ構造を生成する流れ

6 字句解析器の実装

6.1 トークンの設計

構文解析器が先頭のトークンを見るだけでその後の処理をどうすればよいのかを決定できるようにし、煩わしい文字列の処理などを吸収するために以下の 9 つのトークンの種類を用意した。

- String : 空白や改行、記号を含まない文字の列
- Text : 波括弧やダブルクオーテーションで括られたテキストを表す
- Sharp : # を表す
- Comma : , を表す
- Equal : = を表す
- LCurlyBraces : { を表す
- RCurlyBraces : } を表す
- AtMark : @ を表す
- EOF : End Of File の略称で、入力の終端を表す

また、テキストを表す **Text** トークンの実体としては以下の 2 種類のデータのリストとする。これにより、「加工禁止」という指定を上手く反映させることができる。

- BreakableText : 加工することができる箇所の文字列
- UnBreakText : 加工することができない箇所の文字列

さて、具体例を見てみる。

例えば

```
1 @article{id1,
2   tag1 = "text",
3   tag2 = {text {unbreak} breakable},
4   page = 2,
5 }
```

という文字列は以下のリストに変換されることが期待される。

```
1 [
2   AtMark,
3   LCurlyBraces,
4   String("id1"),
5   Comma,
6   String("tag1"),
7   Equal,
8   Text([BreakableText("text")]),
9   Comma,
10  String("tag2"),
11  Equal,
12  Text([
13    BreakableText("text"),
14    UnbreakText("unbreak"),
15    BreakableText("breakable")
16  ]),
17  Comma,
18  String("page"),
19  Equal,
20  Text([BreakableText("2")]),
21  Comma,
22  RCurlyBraces
23 ]
```

6.2 実装

文字列を先頭から読み進めて生成するべきトークンの定義に当てはまらない文字が出現したら「今までの文字列で構成したトークン」と「残りの文字列」のペアを作り、また「残りの文字列」に対して同様に先頭から読み進めてトークンと残りの文字列のペアを作る処理を残りの文字列が無くなるまで繰り返し、トークンの列を生成する。ただし、このとき

- エントリを解析している
- フィールドの中身を解析している
- テキストを解析している

の 3 つ可変の状態を保持し、解析時に先頭の文字の種類と合わせて生成するべきトークンを切り替えていく。実際の文字と状態の種類に応じたトークンの対応は表 1 になる。

表 1: 先頭の文字と状態ごとの字句解析器で生成するべきトークン

	エントリの解析中	フィールドの解析中	テキストの解析中
@	AtMark		
{	LCurlyBraces フィールド解析の開始	Text	UnBreakText
}		RCurlyBraces エントリ解析の開始	フィールド解析の開始
,		Comma	
#		Sharp	
"		Text	フィールド解析の開始
\		Text	アクセントの解析
その他文字列	String	String	BreakableText

さて、SATySFi では文字列を一文字ずつ分割する際には `string-explode : string -> int list` という関数を用いる。この関数は文字列を受け取って Unicode スカラー値の配列を返す関数である。これをパターンマッチと再帰関数を使って解析していく。

例えばフィールドとエントリの解析は以下のような 2 つの再帰関数によって定義することができる。

```

1 %% フィールドの解析を行う関数
2 let-rec lex-field codepoint-lst =
3   match codepoint-lst with
4     %%% whitespace
5     %%% 空白と改行は全て無視をする
6     | x::xs when (9 <= x && x <= 13) || x == 0x20 -> (

```

```
7     lex-filed xs
8
9     %%%% }
10    %%%% RCurlyBracesを生成する
11    %%%% フィールドの終了文字なのでここで処理を終了し、
12    %%%% 生成したトークン配列と共に
13    %%%% 残った文字列も返す
14    | 0x7D::xs -> ([RCurlyBraces], xs)
15    %%%% ,
16    %%%% 一文字だけを読み込んでフィールドの解析に戻る
17    | 0x2C::xs -> (
18        Comma ^:: (lex-filed xs)
19    )
20    %%%% =
21    %%%% 一文字だけを読み込んでフィールドの解析に戻る
22    | 0x3D::xs -> (
23        Equal ^:: (lex-filed xs)
24    )
25    %%%% #
26    %%%% 一文字だけを読み込んでフィールドの解析に戻る
27    | 0x23::xs -> (
28        Sharp ^:: (lex-filed xs)
29    )
30    %%%% "
31    %%%% テキストの解析を行うlex-text関数を
32    %%%% 呼び出し、得られた結果をトークンにする
33    %%%% 残った文字列に対してlex-filed関数を適用して
34    %%%% フィールドの解析に戻る
35    | 0x22::xs -> (
36        let (text-lst, xs) = lex-text `"\` 0 [] String.empty xs
37        in
38            (TextList text-lst) ^:: (lex-filed xs)
39    )
40    %%%% {
41    %%%% テキストの解析を行うlex-text関数を
42    %%%% 呼び出し、得られた結果をトークンにする
43    %%%% 残った文字列に対してlex-filed関数を適用して
```

```

43     %%% フィールドの解析に戻る
44     | 0x7B::xs -> (
45         let (text-lst, xs) =
46             lex-text `}` 0 [] String.empty xs
47         in
48         (TextList text-lst) ^::: (lex-filed xs)
49     )
50     %%% 自然数の解析を行う
51     %%% 自然数を取り出せたら文字列に変換して
52     %%% テキストとして扱う
53     | x::xs when 0x30 <= x && x <= 0x39 -> (
54         let (n, xs) =
55             lex-int (string-unexplode [x]) xs
56         in
57         (TextList [Text(n)]) ^::: lex-filed xs
58     )
59     %%% 空白文字や記号が出現するまで文字を
60     %%% バッファにためるlex-string関数を呼び出し、
61     %%% 文字列を生成させ、その結果をStringトークンに入れる
62     %%% 残りの文字列に対してlex-field関数を再度適用して
63     %%% フィールドの解析に戻る
64     | x::xs -> (
65         let (s, xs) =
66             lex-string String.empty codepoint-lst
67             in
68             if String.equal s String.empty then
69                 abort-with-message (
70                     `Error: Unexpected character " `#
71                     ^ string-unexplode [x] ^ `` `#
72                 )
73             else
74                 (String s) ^::: (lex-filed xs)
75         )
76         | _ -> abort-with-message `Error: Unexpected EOF`
```

77

78 %% エントリの解析を行う

79 let-rec lex-entry codepoint-lst =

```
80  match codepoint-lst with
81    %%% whitespace
82    %%% 空白と改行は全て無視をする
83    | x::xs when (9 <= x && x <= 13) || x == 0x20 -> (
84      lex-entry xs
85    )
86    %%% @
87    %%% 一文字だけを読み込んでエントリの解析に戻る
88    | 0x40::xs -> (
89      AtMark :: (lex-entry xs)
90    )
91    %%% {
92    %%% LCurlyBracesを生成し、
93    %%% 残りの文字列についてフィールド
94    %%% の解析を行うlex-filed関数を呼び出し
95    %%% lex-filed関数が生成されたトークン列と合わせる
96    %%% lex-filed関数が処理した後の残った文字列について
97    %%% lex-entry関数を再度適用し、エントリの解析に戻る
98    | 0x7B::xs -> (
99      let (token-lst, xs) = lex-filed xs in
100     List.append (LCurlyBraces :: token-lst) (lex-entry xs)
101   )
102   %%% 空白文字や記号が出現するまで文字を
103   %%% バッファにためるlex-string関数を呼び出し、
104   %%% 文字列を生成させ、その結果をStringトークンに入れる
105   %%% 残りの文字列に対してlex-entry関数を再度適用して
106   %%% エントリの解析に戻る
107   | x::xs -> (
108     let (s, xs) = lex-string String.empty codepoint-lst in
109     if String.equal s String.empty then
110       abort-with-message (
111         `Error: Unexpected character ``#
112         ^ string-unexplode [x] ^ ```
113       )
114     else
115       (String s) :: (lex-entry xs)
116   )
```

117

| [] -> [EOF]

文字列をバッファに入れて文字列を生成するための `lex-string` 関数やテキストの解析を行う `lex-text` 関数についても同様に、先頭の文字の値に応じて行う処理を切り替えつつ再帰関数で処理を行う実装となる。

7 構文解析器の実装

7.1 出力するデータ構造の設計

トークンの列を解析して木構造や配列などのプログラムが扱いやすい構造に変換するのが構文解析器の役割である。このデータ構造は元のフォーマットが表している構造、例えば Key-Value 形式であるとか入れ子構造であるとかを表すようなデータ構造にするのが最も自然な形である。

ここで BibTeX フォーマットをもう一度見てみる。

```

1 @article{id1,
2     tag1 = "text",
3     tag2 = {text},
4     page = 2,
5 }
6
7 @misc{id2,
8     tag1 = "text",
9     tag2 = {text},
10 }
```

すると、大まかに

- ID を Key とし、article などの掲載場所の情報とフィールド情報のペアを Value とする Map 構造
- タグを Key とし、テキストを Value とする Map 構造

のような構造が見えてくる。つまり前述の BibTeX 情報は

```

1
2 "id1" - (
3     "article",
4     [
5         "tag1" - "text",
```

```
6     "tag2" - "text",
7     "page" - "2",
8   ],
9 ),
10
11
12 "id2" - (
13   "misc",
14   [
15     "tag1" - "text",
16     "tag2" - "text",
17   ]
18 ),
```

のような疑似的なコードで表すことができるわけである。これを SATySFi の型として表すと

```
1 type text-token =
2   | BreakableText of string
3   | UnbreakText of string
4
5 %%% (|
6 %% key : value;
7 %% |)
8 %% という記法は struct やレコードと一般に呼ばれるもの
9 type entry = (
10   entry-type: string;
11   fields : string (text-token list) Map.t;
12 )
13
14 type bibtex = string entry Map.t
```

見慣れない表記かもしれないが、概ね雰囲気は伝わるだろう。

7.2 実装

構文解析器の実装も字句解析器と同様にトークンの列を頭から読み進めて生成するべきデータ構造の定義に当てはまらないトークンが出現したら「今までのトークンで構成したデータ」と「残りのトークン列」のペアを作り、また「残りのトークン列」に対して同様に

先頭から読み進めてデータ構造と残りのトークン列のペアを作る処理を、トークン列が無くなるまで繰り返し最終的に一つの大きなデータ構造を生成する、という手順で行う。

BibTeX フォーマットではテキスト同士の結合やテキストの定義とその参照ができるようになっており、それらの処理をする必要があったり、コメントやテキスト定義用のエントリが専用にあったりするが、以下に例示するコードではこの処理は省略する。

```

1  %%% `tag = {text}` の 解析 を 行う
2  %%% カンマ が あつたら parse-field 関数 を
3  %%% 再度 適用 して 解析 の 続き を 行う
4  %%% カンマ が 無かっ たり `}` が あつたら
5  %%% 処理 を 終了 する
6  let-rec parse-field map token-lst =
7      match token-lst with
8          | (String key)::(Equal)::(Text text)::xs -> (
9              match xs with
10                 | (Comma)::xs -> (key, TextList(text)) ^:: parse-field
11                     xs
12                 | (RCurlyBraces)::_ ->([(String.trim key, TextList(text))], token-lst)
13             )
14             | (RCurlyBraces)::xs -> ([] , token-lst)
15             | x::_ -> abort-with-message (`Error: Unexpected token `#
16                 ^ token2message x)
17             | [] -> abort-with-message `Error: Unexpected EOF`
18
19  %%% `{cite-id}` 部分 を 取り出 して 参考文献 に 振られる 一意 の ID を
20  %%% 取得 する
21  %%% 取得 できたら parse-field 関数 に 残り を 解析 させ、
22  %%% 最後に `}` が 出現 す ること を 確認 して 終了 す る
23  let parse-entry entry-type token-lst =
24      match token-lst with
25          | (String cite)::(Comma)::xs -> (
26              let (lst, xs) = parse-field xs in
27              match xs with
28                  | (RCurlyBraces)::xs -> (
29                      let f (key, tok) map =

```

```
28         match tok with
29         | TextList(l) -> Map.bind (key |> String.lowercase-
30             ascii |> String.trim) l map
31         | x -> abort-with-message (`Error: Unexpected token
32             `# ^ token2message x)
33     in
34     let map = List.fold Map.empty f lst in
35     let bib =
36       (|
37         entry-type = entry-type;
38         fields = map;
39         |)
40     in
41     (cite, bib, xs)
42   )
43   | x:::_ -> abort-with-message (`Error: Unexpected token
44       `# ^ token2message x)
45   | [] -> abort-with-message `Error: Unexpected EOF`
46   )
47   | x:::_ -> abort-with-message (`Error: Unexpected token `#
48       ^ token2message x)
49   | [] -> abort-with-message `Error: Unexpected EOF`  

50  

51 %%% `@article{`部分が出現したらその後エントリ内部の処理に移
52   る
53 %%% データが取れたらMapに追加し、parse-bibtexを呼び出して
54 %%% 同様に処理をする
55 %%% ファイルの終端が現れたら終了
56 let-rec parse-bibtex token-lst =
57   match token-lst with
58   | (AtMark)::(String s)::(LCurlyBraces)::xs -> (
59     let s = s |> String.lowercase-ascii |> String.trim in
60     let (cite, entry, xs) = parse-entry s xs in
61     Map.bind (cite |> String.trim) entry (parse-bibtex xs)
62   )
```

```

60 | [EOF] -> Map.empty
61 | x::_ -> abort-with-message (`Error: Unexpected token `#
  ^ token2message x)
62 | [] -> abort-with-message `Error: Unexpected EOF`
```

さて、これで字句解析器で生成したトークン列を `parse-bibtex : token list -> bibtex` 関数に与えることで BibTeX フォーマットを表すデータ構造を得ることができるようになった。

8 API の設計

内部的な設計の変更や意図しない利用方法に対応するためにはデータ構造の具体的な実装はモジュールの中に隠蔽し、操作するための API だけを露出させるのが好ましいとされている。特に今回は「加工禁止」のテキストが存在するため、ユーザーが弄ることができる範囲を制限する必要があった。

まず、値を取得するための関数として

- 参考文献に振られる一意の ID を指定すると参考文献全体のデータから一つの参考文献情報を取得する関数
- フィールドの Key を指定すると一つの参考文献情報からテキストを表すデータを取得する関数

の 2 つを提供するという構成とした。これにより、たとえば「tanaka-2022 という ID の参考文献のタイトルを取得する」という場合は

```

1 bibtex-str %%% BibTeX フォーマットの文字列
2 |> parser
3 %%% IDを指定して一つの参考文献情報 (entry型) を取得する
4 |> get-entry `tanaka-2022`
5 %%% タイトルを取得する
6 |> get-text `title`
```

のように書く、というインターフェースが実現される。ユーザーが具体的なデータの中身に触ることはできないが、それでも問題なく所望のデータを取得できることがわかる。

データの取得はこれでできるようになったが、このままでは文字列の取得に関しては課題が残る。現状のインターフェースでは BibTeX データで指定されている「加工禁止」の指定を守ることができておらず、加工可能な部分と加工が禁止されている部分も一緒くたにしてユーザーに値を返してしまっており、そこを遵守するかどうかなどがユーザーに任せている。そこで、`get-text` 関数を拡張し、「テキストに対してどのような加工を行うのか」や「どの文字で分割するのか」を指定できるようにした。たとえば、「著者情報を取得して‘and’で分割し、先頭を大文字化する」という処理を行うためには

```
1 bibtex-str %%% BibTeX フォーマットの文字列
2 |> parser
3 %%% IDを指定して一つの参考文献情報(entry型)を取得する
4 |> get-entry `tanaka-2022`
5 %%% タイトルを取得し、分割し、先頭を大文字化する
6 |> get-text `author` (Some `and`) capitalize-ascii
```

というコードを書くインターフェースとなる。内部の処理では加工禁止文字列をうまく回避するように複雑な関数の適用を行い、結合の処理を行っている。

9 使い方

表示したい参考文献の ID を受け取って該当する参考文献情報についての

- 著者名のリスト
- 小文字化したタイトル
- 出版年

の 3 の情報を取得して並べるコマンドを作るには次のように実装すれば良い。

```
1 let-inline \demo id bib =
2   %%% IDを指定して参考文献情報を取得
3   match get-entry id bib with
4     | Some(entry) -> (
5       let authors =
6         %%% `author`で取得して`and`で分割
7         %%% それ以外の加工は行わない
8       let author-lst-opt =
9         get-text `author` (Some `and`) identify entry
10      in
11      match author-lst-opt with
12        | Some(l) -> (
13          %%% コンマ区切りで並べる
14          let s = String.concat ?:(`, `#) l in
15          embed-string s
16        )
17        %%% authorがなかったらパニック
18        | None -> abort-with-message `err`
```

```

19   in
20   let title =
21     %%% `title`で取得して分割しない
22     %%% 全部小文字化
23   let title-opt =
24     get-text `title` None lowercase-ascii entry
25   in
26   match title-opt with
27   | Some([s]) -> embed-string s
28     %%% titleがなかったらパニック
29   | _ -> abort-with-message `err`
30   in
31   let year =
32     %%% `year`で取得して分割しない
33     %%% 加工は行わない
34   let year-opt =
35     get-text `year` None identify entry
36   in
37   match year-opt with
38   | Some([s]) -> embed-string s
39     %%% yearがなかったらパニック
40   | _ -> abort-with-message `err`
41   in
42   {#authors;. "#title;". #year;.}
43 )
44 %% 該当する参考文献情報が見つからなかったらとりあえずパニック
45 | _ -> abort-with-message `err`
```

実際に

```

1 @Article{田中2022,
2   Author = {田中 さとう and 佐藤 たなか},
3   Title = "世界の「田中」論",
4   Publisher = "Addison-Wesley",
5   journal = "苗字学会会誌",
6   year = year_1994
```

```
7 }  
8  
9 @BOOK{Lamport94,  
10   Author = {Leslie Lamport and {L and Lamport 2}},  
11   Title = {{\LaTeX}: A Document Preparation System},  
12   Publisher = "Addison-Wesley",  
13   pages = "156-163",  
14   year = 1994  
15 }
```

という BibTeX フォーマット文字列に対して

```
1 \demo (`Lamport94`) (parser bibtex-str);
```

とコマンドを適用させてみると、図 4 のようになり、意図した挙動をしていることがわかる。実際に文書に組み込む際には、より適切なデザインになるように取得する情報を増やすたり記号を変えたりする必要があるが、概ねこのような形で簡単に実装することができる。

Leslie Lamport , L and Lamport 2. "LaTeX: A document preparation system". 1994.

図 4: デモの実行結果

10 おわりに

SATySFi で BibTeX フォーマットの解析を行うライブラリを作成することができた。これで SATySFi で参考文献情報を取り扱うことがより容易になり、アピールポイントにもなることを期待している。今後は個別の参考文献表示ライブラリの要求するデータ構造に合わせて変換するライブラリを作ったりするなど、エコシステムの充実化をしていきたい。

ここで実装したライブラリは GitHub のリポジトリ (<https://github.com/puripuri2100/SATySFi-bibtex-parser>) で MIT ライセンスの下公開しているため、自由に触ってみてほしい。SATySFi のパッケージマネージャへの登録は諸事情によりまだできていないが、その諸事情が解消され次第登録する予定である。

SATySFi でのプログラミングの雰囲気を感じてもらい面白いと思っていただければ、また構文解析の面白さやライブラリの作り方などについても伝えることができていれば幸いである。

バカとブスこそインターンに行け！

文 編集部 maetin

1 はじめに

バカとブスこそインターンに行け！（特定の団体・個人を批判するものではありません）

「coins に入ったはいいけど授業でやるプログラミングは案外少ないし、自分で作りたいものも興味ある技術もまだわからないよ～;;」という coins 生は案外多いのではないか？ そんな coins 生にとって、インターンはうまく活用すると大きいメリットがあるでしょう。今回はインターンを通して得られるメリットと、インターンを探す方法を紹介します。

本記事で述べる「インターン」は基本有給の就業型インターンであることをまず断っておきます。また、本記事は自らで課題発見・課題解決ができる実装型人間を対象にしています。

2 なぜインターンに行くといいのか

2.1 インターンで得られる経験・スキル

AC 入試などで入学した同期をみて、「自分はあんなに好奇心を持って自分で開発するのは難しいな……」と思うのは coins 生の中では案外多いのではないでしょうか？ 実際自分が興味を持った技術を主体的に学んでいき、独力で様々なものを作り上げるのは多くの人ができることではないでしょう。

そういった個人開発を積極的にできない coins 生にとって、インターンはとても有効な学習手段です。インターンでは労働契約の下に責任が発生するため、強制的に自分で学習する必要が生じます。そういった外部からの圧力によって、講義以外でも様々なことを学ぶモチベーションが生まれると筆者は考えています。インターンで実際に経験できる可能性の高いことをいくつか挙げてみます。

- チーム開発の手法（Git, ドキュメント管理, PR レビュー等）
- Web アプリケーション開発（フレームワーク, データベース, インフラ等）
- 株式会社の雰囲気
- プログラミング以外のソフトスキル（コミュニケーション, プрезентーション, ビジネスマナー等）

これらの経験は、受動的に大学の講義を受けているだけでは中々得られないものです。また、副次的に（人によってはこちらがメインになるかもしれません）お金を稼ぐこともできます。

2.2 筆者のインターン経歴

筆者がこれまで経験してきたインターンは三社あります、以下にその内訳を紹介します。

- 1 社目：従業員数 100 名前後・年商数十億の HR 系ベンチャー (2022/08 - 2023/06)
- 2 社目：従業員数 1000 人以上・東証プライム上場の営業 DX SaaS 企業 (2023/08 - 2023/09)
- 3 社目：従業員数十数人の医療系ベンチャー (2023/10 -)

創業期からちょうど伸びている時期、伸び切った時期の企業のいい面・悪い面をそれぞれ経験できたのでそういった点も含めてインターンのメリットを紹介していきます。

創業期の企業

創業期の企業は、まだまだ社員数が少なく、社内のルールもまだまだ整っていないことが多いです。そのため、自分で考えて行動する必要のある場面が多く、自分の成長に繋がることが多いです。また、社員数が少ないため学生でもしっかりととした根拠を持って話せば自分の意見を通すこともしやすいです。

しかし、社員数が少ない（フルタイムエンジニアの数が 5 人未満）ということは社内のエンジニアの人に色々指導してもらうということが難しい傾向にあり、かなり自分で努力・行動する必要があります。

伸びている時期の企業

伸びている時期の企業は、社員数が多くなってきていため創業期の企業に比べて社内のルールも整ってきていることが多いです。またある程度土台が出来上がっているプロダクトに関わる可能性が高いので、すでに実装されているコードを読むことで勉強になる面もあります。それに加え、創業期の企業のような柔軟な社内文化も残っているため、大きい機能開発を任せられることや、自分が主体性を持って業務プロセスの改善などを行うことも可能でしょう。

しかし、まだ利益が完全に確立できていない場合は給与水準が低くなる傾向があり、会社の身内感も強いためやりがい搾取のような形態になってしまふこともあります。

伸び切った時期の企業・大企業

従業員数が 500 名を超え、売上も安定している時期の企業では外部コンプライアンスなどの点から社内のルールは明文化され、社内の文化も固まっていることが多いです。

そのため新人に対する教育体制が整っていることが多く、運用されているプロダクトも巨大なため非常に多くのことを学ぶことができるでしょう。また給与の面においても学生とは思えないような高額な給与を頂ける場合が多く、扶養上限まで働いても自分の時間を多く取れます。

殆どの場合デメリットは存在しませんが、採用されるハードルがかなり高くなっていることが多いです。

2.3 就活におけるインターンのメリット

就活における就業型インターン経験のメリットは非常に大きいです。会社での働き方をそのまま経験している分、非常に具体的なエピソードを話せるようになります。そのため採用面接においてはかなり感触よく受け取ってもらえることが多いです。またそのままインターン先で内定をもらえることも多く、その場合就活の手間は大きく軽減されることとなります。

しかし、就活が面倒くさいからとあまり大手の選考に出さずにインターンで就労しているベンチャー企業にそのまま内定をもらうというのは、キャリアや給与など多くの点で不利になる可能性があるので、仮に内定が出たとしてもある程度別口で就活をする必要はあるでしょう。

1-2年次からインターンをすると良い

就活においてインターンはかなり有用であると述べましたが、そのためには就活が始まる時期よりも前にインターンを経験しておく必要があります。現代の就活では3年の夏のインターンから選考が始まっていることが多く、インターンの選考は4月から6月にかけて行われます。

そのためインターン経験を活用した就活をしたい場合、1-2年次の段階でインターンを経験しておく必要があります。また、大学院に進学する場合でも3年次の段階で「自分は学部就職します！」という体で大手のインターンに参加することで、院進後の本番の就活で有利になるでしょう。

次章以降では1-2年次でインターンを探す場合・就活のタイミングでインターンを探す場合に分けて、それぞれの探し方を紹介していきます。

3 インターンを探す方法

3.1 1-2年次でインターンを探す場合

まず最低限の開発経験を積む

いくら人手が欲しいと考えている企業でも、for文から教えるコストを払うつもりのある企業は少ないでしょう。そのため、最低限のプログラミング知識は身につけておいたほうが最初のインターン先を見つけるのが楽になります。具体的には、

- 競技プログラミングをやる
- 学内向け・個人向けの簡単なWebアプリケーションを作る
- FlaskやDjangoなどのWebフレームワークを触ってみる

などが初めの開発体験としては比較的取り組みやすいと思います（諸説あり）。

技術マッチョの友達を作る

初めの経験を積む際や、インターン先の条件を相談する際には、技術マッチョの友達を作っておくと非常に便利です。技術マッチョの人間は、AC入試で入った人やWORD編集

部にいる人に多いです。技術マッチョの友達を作るメリットとしては、技術の様々なトピックの取っ掛かりを教えてもらえることや、やりたいことを相談することで自分のやりたいことを明確にすることができることが挙げられます。また、技術の話をすることの面白さを早い段階から知っておくと後々の開発体験が大きく変わると思います。

インターンを探す

1-2年次でインターンを探す場合、選考の一部として募集をかけているインターンではなく、労働力・バイトとしてインターン生を求めている企業の募集に応募した方が採用率が高いです。そのため、対象とする企業の規模としては従業員数100名前後、またはそれ以下の企業を狙うのが良いでしょう。

インターンを探す方法としては

- ゼロワンインターン^{*1}
- Infra インターン^{*2}

などのインターン専門求人サイトなどで探すのが一番手っ取り早いです。

また、友人がインターンをやっている場合は友人に紹介してもらうと探す手間も省けるので大変楽です。

4 3年次以降にインターンを探す場合

3年次以降でインターンを探す場合、新卒選考の目的を孕んだインターンが多くなります。これまでにインターン経験・または課外での開発などの経験がある場合はそのまま大手の選考目的インターンに参加するのも手ですが、そうでない場合は1-2年次のときに述べたようなインターン経験を積むところから始めたほうが良いことが多いです。大学院に進学することを決めている人でも、3年次の段階で新卒採用のインターン選考を経験しておくと、就活のスケジュールや人事面接が何を求めているかをある程度知ることができます。どちらにせよ3年で色々なところへインターンの応募をすると良いでしょう。

5 まとめ

インターンは大学生にとって非常に有用な学習手段であることがわかりました。インターンを意識的に、積極的に活用することで就活や個人開発においても大きなメリットを得ることができます。

ぜひインターンを活用して、自分の成長の機会を増やしていきましょう。

*1 <https://01intern.com/>

*2 <https://www.in-fra.jp/>

大学内で世界一臭い缶詰を合法的に開けた話

文 編集部 cely chan

こんにちは。WORD 編集部員の cely chan です。最近 AP を買いまくってます。たっけえ。しかも、買ったやつがうまく使えなくて、返品されていきました。悲しい。『WORD は大学新聞の学内誌特集からハブられました号』で博士号回収記録とかいう記事を書きましたが、今回は博士号関連ではなく、『大学内で世界一臭い缶詰を合法的に開けた話』と題して、T-ACT 企画承認番号 22022A 世界の食文化に触れてみよう会の裏話を寄稿いたします。なお、正規の成果報告書はそのうち、T-ACT の活動報告書として、年度末を目処に発行されるはずですので、本記事はあくまで、副読本程度に捉えてください。また、本記事に書けなかったこともありますので、ご興味をお持ちになりましたら、直接ご連絡をいただければと思います。

1 大学で開けることになった経緯

世界一臭い缶詰を大学内で開ける必要に迫られたのは、T-ACT の公式ページにある活動目的^{*1}に書いてある通りの理由である。以下は、その活動目的である。長いので、読み飛ばしていただいて問題ない。

筑波大学は、建学の理念にある「国内的にも国際的にも開かれた大学」として、さまざまな取り組みをおこなっている。国内向けとして、学内に設置された各センターは共同研究を積極的に受け入れており、また外部組織との連携をおこなっているセンターも数多い。授業についても、科目等履修生制度による外から中を開かれた構造及び、法人細則に基づく、他大学等における授業科目の履修制度による中から外に開かれた構造を持っている。海外向けについては、68 カ国・地域の大学や研究機関と 378 協定を 2022 年 12 月 22 日時点で締結しており、またスクーデンツサポートセンターが留学相談を親身におこなってくださるなど、中から外への充実した開かれた構造を持っている。

しかしながら、私が生活する限りは、海外向けについて外から中に向かた構造、つまりは海外の文化を学内に持ち込む構造が最も充実していると感じる。例えば、積極的な留学生の受け入れをおこなっており、全学生約 15,000 人に対して 2000 人近くの留学生が学びを深めている。そのため、筑波大学には留学生をメンバーとする団体が複数存在し、留学生と日本人学生の文化交流を主とした団体も存在している。また、文化の根幹をなす食についても充実しており、ハラルに対応した食堂が 2 店舗あることに加え、アフリカ料理をはじめとした、海外の郷土料理を提供するキッ

*1 <https://www.t-act.tsukuba.ac.jp/project/view/?seq=601>

チンカーが頻繁に出店しており、異国の文化を理解しやすいと考えている。これらの構造により、大学内は国際性が十分に日常化し、豊かな人間性を育むに相応しい場所になっていると考えられる。

ところがこの構造で提供される異国文化は、大学などがある程度フィルタリングを行った、誰に対してもほとんど害のない文化であり、これらを吸収したところで、真の理解を獲得することは難しい。この構造を例えて言うならば、和食の紹介の際に味噌汁や御節料理を紹介し、納豆やくさや、イカの塩辛など、文化に根ざし地域性をもつが、その特徴ゆえに回避されがちな食材を紹介しない行為に等しい。このように、誰に対しても好まれる海外文化のみを吸収していくは、海外文化には好き嫌いが分かれる要素がないという偏見を持つ可能性がある。また、その文化の真髄に迫りうる要素を見落とす可能性があり、真の理解を獲得することが困難となる。この状態を回避するためにさまざまな調査を行おうとするも、特徴的で回避されがちな食品は国内で入手することが難しく、他者の経験をもとにした知識を蓄えることで対応せざるをえなくなる。この対応にも問題があり、他者の意見を鵜呑みにすることで、他人の経験によるバイアスを無視したり、経験した他者の個人的な意見を忘れたりすることがある。また、他者の経験をもとに議論をしていくは、自己の考えを育てることができず、創造的な知性を育むことは困難になる。

そこで本活動では、スウェーデンの文化に根ざしたシュールストレミングを実食し、なぜ国外で敬遠されるこの食材が現地で食べられているのかについて考察することで、真の国際理解と多面的な視点を持つ創造的な知性の育成を始めるきっかけを提供する。また同時に、匂いの原因であるガスの成分を分析することにより、なぜ特有の匂いが発生するのか考察する機会も提供する。

実際には、この活動目的に加えて、缶詰の消費期限が過ぎようとしていたため、開封を決意した。

この缶詰は、基礎体育野外運動において親密になった集団が、2021年の年末に闇鍋を計画し、その際に利用しようとしていたものである。しかしながら、室内での開缶となりうこと、参加者からの激しい抵抗などの理由から開封は見送られ、大学内某所にある冷蔵庫内で、一年ほど封印されていた。一般に缶詰は保存食品としての傾向が強いため、その消費期限は長くなりがちであるが、この食品は加熱をしていないため、消費期限は製造から1~2年程度であった。製造、スウェーデンからの輸送、日本での保管などを考慮すると、購入してから1年ほど経った2022年の年末、2023年の年始は、ちょうど消費期限なのである。

開封せず永久に封印することや、そのまま廃棄物として廃棄することも考えたが、永久に封印したとしても、微弱な発酵を止めることはできず、いつしか臨界²に達すること。及び、ゴミ収集車内で爆発することの危険性を考慮した結果、開封して実食する運びとなつたのである。

*2内圧上昇に伴う爆発のこと

2 大学で開ける際に必要な政治

開封場所をどこにするか、非常に迷った。シュールストレミングは非常に臭いらしいという予備知識はあったので、ひらけた場所を用意する必要がある。ただ、大学近辺のひらけた場所を利用すると、近隣住民から苦情が飛んできて警察を呼ばれる可能性が高い。これを避けるために人気がないところに行こうと自動車を使った場合、行きはなんとかなるが帰りの車内が悪臭まみれになると予想される。第三者に迷惑をほとんどかけず、かつ広い場所はどこかないと悩んだ結果、大学内の石の広場が最適であると判明し、これを利用することになった。

大学内で何をしようが学生の勝手であると思われるかもしれないが、実はそうではなく、様々な制限がある。また、大学内で異臭騒ぎとなれば大事になりうるし、警備員や学生生活課がすっ飛んできて活動の邪魔をすることが予想される。そのため、周囲からの抵抗を受けることなく、正当かつ安全に缶詰を開けるよう、大学の公認企画として実施し、有識者の知見を活用することを思いついた。ここで、学生のやりたいことを、大学として支援してくれる T-ACT の利用を閃いたのである。

T-ACT は、大学組織の一部がお墨付きを与えた企画であるので、開催中にやいやい言わ�る心配は非常に低くなる、その代わりに、企画を実施するために多くの紙を書き、会議を通す必要がある。プランナー(首謀者)以外に、オーガナイザ(協力者)、パートナー教員(参謀)の 2 名が必要である。今回は、T-ACT を利用してアヒル・ボートを松美池に浮かべる企画を行った非常勤教員にパートナー教員を、闇鍋参加者のうち 1 人にオーガナイザーをそれぞれお願ひし、企画を提出した。

3 大学で食べる際に必要な政治

世界一臭い(とされる)缶詰を開ける際に必要となる政治と、世界一臭い(とされる)缶詰を食べる際に必要となる政治は、同じ大学企画であったとしても、かなり異なっている。前者は単に開けるだけであり、企画として提案できる人数がいるのか、異文化の尊重について十分配慮しているかの注意こそ必要^{*3}であるが、それ以外については、微々たる問題しかない。しかしながら、食べるとなると、難易度は飛躍的に高まる。なぜなら、開封・盛り付けは調理であり、食中毒対策として、衛生管理が必要となるためである。

T-ACT は大学の公認企画であるため、衛生面の配慮について、雙峰祭の飲食企画と近いレベルで管理を行う必要がある。調理者の健康診断等は不要であるが、保健所に対して届出を行い、衛生面で問題がないかを確認した上で、T-ACT の審査会のようなものに提出し、先生方の認可を受ける必要がある。

この認可を通すために、私は保健所に突撃して紙を提出したり、学生生活課に連絡してテントを借りたり^{*4}していた。この作業におよそ 2 週間ほどかかったが、授業の合間に 1 人でこなすことのできる程度の難易度であった。

*3 安っちは YouTuber などがやっている、とりあえず開けてみて臭かったです！ こんなのが食べるなんて信じられない！ みたいな、しょうもなく、かつ他者への思いやりや想像力を著しく欠いた言動をしないこと

*4 屋根がないところでの調理は、一般的によくないこととされているので、屋根を創成する必要があった

4 大学における衛生管理の真実

前述の、「大学で食べる際に必要な政治」をこなす過程において、筆者がおもしろいと感じる事実が判明したので、以下にその内容を共有する。なお、一部を伏せ字とする。

これは、大学という教育機関の特性を考えれば当然であるが、よほどのことがない限り生徒個人に責任を押し付けてこなかった高校生活や、ただただぼんやりし続け、口へ運ばれる授業を消化してきたこれまでの大学生活とは大きく違い、大学生としての責任や、大学と学生の関係について、急に突きつけられる形となったので、大変におもしろく、またびっくりした。

5 保険を用いた責任回避の甘さ

前述のこの責任問題を回避するために、とりあえずということでレクリエーション保険に加入したが、ここで、パートナー教員より指摘があり、「とりあえず加入したレクリエーション保険について、食中毒がこの保険でカバーされるのか、きちんと約款を取り寄せて、弁護士と一緒によく読まねば断言できない。食中毒が判明したタイミングや、本企画との因果関係について明確にしておかねば、言いくるめられて、こちらが泣きを見ることになる」とのご意見をいただいた。確かに、何も考えずに「とりあえず保険に入っておけばええやろ！」の思想はあまりにも楽観的であり、また無意味を超えて有害な行為にもなり得る。そのため、T-ACT の予算より弁護士相談費用を出していただき、以下の内容について質問をした。なおこの質問に際して、不法行為やその責任について、ある程度民法の資料を読み込んだ。

1. 保険によってカバーされる食中毒は、急激かつ偶然な外来の事故のみであり、後日起こる型の食中毒については補償されるか不透明である。
 2. 自己責任であることを確認する同意書について、消費者契約法 8 条 1 項により無効になる可能性がある。
 3. 死亡した場合の保険金額が少なく、その免責条項そのものが信義則に反して無効と

*5責任の取りようがないといえば、確かにそうである。



図1：廃棄箱

される可能性がある。

6 大学で後片付けする際に必要な政治

シュールストレミングは、世界一臭いとされる缶詰なだけあって、それなりに強い悪臭を発生させるのである。また、生魚の塩水漬け^{*7}であるため、缶内で発酵が進み、酪酸やプロピオン酸などの有機物が生成されている可能性がある。ここで、筆者が面白半分でとった^{*8}、事例に学ぶ環境安全衛生と化学物質(1414014)において、「実験で生じた廃液などを直接流しに捨てると、各建物下にある下水貯留槽の内容物を全て吸い出す必要に迫られる^{*9}ので、廃液などはきちんと処理をせよ」という指示があったことを思い出し、廃棄物関連の大学部署に連絡をとった。すると、なんだかヘンな方がいらっしゃるようで、パッキン付きの実験廃棄物(産業廃棄物?)処理容器を一箱ご提供いただき、これに全てを廃棄できることになった。

*⁶あまりにも元も子もない意見であるため 黒塗りとします すみません。

*⁷ 加熱処理をしていないので、厳密には年鑑ではないのである。

*8面白半分で授業をとりままではいはない

*9 実験系の排水を下水に流してしまうと、下水処理工場に影響を与える + すごい額の請求が飛んでくる可能性があるため、各エリアの地下に「下水のバッファーやいた駆除槽があり、PHなどを駆除している」

7 おわりに

本記事では、大学内で世界一臭い(とされる)缶詰を開ける企画の裏側を記述した。本記事には書ききれなかったおもしろ話(附属病院調理室なら衛生上安全だろうし、ここで調理してもらおうと調整した等)や、黒塗りとなってしまった部分の中身がまだいくらかあるので、興味のある方は筆者までご連絡いただきたい。

また、筑波大学の T-ACT は非常に素晴らしい企画であり、何かやりたいことがある学生は、とりあえず T-ACT フォーラムに行くことをお勧めする。人的・物的資源について、大学から手厚い支援を受けることができ、自分のアイデアをより高めることができる。

メールバトル 2023

文 編集部 raspi0124 & momeemt

こんにちは。学園祭実行委員会情報メディアシステム局の^{*1}raspi0124 と momeemt です。多くの学生団体では現実的な選択としてメールアドレスや Google アカウント、その他サービスアカウントの共有が行われると思います。この記事では筑波大学学園祭実行委員会のメールをセキュアな構成にするための道のりと、それに伴い現在進行形で起こっている揉め事^{*2}についてご紹介します。

1 経緯

まず、移行以前の環境について説明します。

学園祭実行委員会（以下、学実委）では学生団体や大学、協賛企業様などとのコミュニケーションにメールを利用しています。また、「sohosai.com」というドメインを保有しており、これを利用してメールを送るために、Google Workspace Business Standard を 6 アカウント分契約していました。それぞれのアカウントは総合窓口用、本祭における企画団体とのコミュニケーション用、協賛企業様とのコミュニケーション用、などざっくりと役割が分けられており、メールに関わる委員が各々 Google アカウントにログインして利用していました。

つまるところ各局、各委員が同じアカウントにパスワードを共有してログインしており、かつその方法については定められていないという状態です^{*3}。実委の代替わりが 1 月上旬に起こりましたが、当時はそれをかなり問題に感じていました^{*4}。もし 400 万円もの自由に使えるお金があれば、全員分の Google Workspace アカウントを契約することで解決できるのですが、残念ながらありませんでした。そこで、Google Workspace Business Standard の利用をやめてメールアドレスが無制限に発行できるメールサーバに移管したいと考えるようになりました。

2 検討

利用するメールサーバには以下の選択肢が挙げられました。

- さくらのメールボックス
- ムームーメール

^{*1}名乗っておいて申し訳ないのですが以下の文章は個人らの見解ですのでご理解ください。

^{*2}他人事のようですが、人と大揉め事を起こしても仕方がないので、貴重な大学生活の時間をどのように使うかはよく考えましょう。

^{*3}パスワードの共有には Bitwarden の Send リンクの発行が行われることが期待されますが、実際には Slack の DM、LINE のトークやノートなどで共有されていたであろうことは容易に想像できます。

^{*4}血気盛んだった

- カゴヤ・ジャパン メールサーバー
- セルフホスト

まずセルフホストを将来的に維持していく未来が見えなかつたので除外します^{*5}。予算が限られているので主に料金面、発行できるメールアドレスの数などを重視して、さくらのメールボックスを選択しました。さくらのメールボックスは最小構成で年額 1048 円という破格の安さ^{*6}で、メールアドレスも無制限に発行できます。学実委はさくらインターネット株式会社様にご協賛をいただきており、技術的な相談もできるかもしれないと思ったことも理由の 1 つです。

2.1 メールクライアント

メールクライアントの選定にあたつては概ね以下の要件を元に検討しました。

- 基本的なメールクライアントとしての機能が備わっていること
- セルフホストが可能であること
- 無料で使用できること
- Web 上で扱えること
- 赤入れ機能や上長承認機能が備わっているか、プラグインを用いた機能付加システムが備わっていること
 - そのドキュメントが整備されていること、もしくはインターネット上に一定の資料が存在すること
- 2023 年に入つても開発が継続されていること
 - ある程度のユーザ数・コントリビュータ数があり、これからも開発がある程度継続される見込みがあること

こうした要件をほとんど満たすクライアントは意外なほど数少なく、ざつと調べた限りでは僅かに Cyphr や RoundCube などが挙げられるのみでした。Cyphr は基本的なメールクライアントとしての機能^{*7}を備え、かつセルフホストが可能で無料で使用可能、かつ 2023 年に入つても開発が継続され、プラグインについても一応システムとしては存在するものの、プラグインを開発するにあたつてのドキュメントが Roundcube と比べて整備されていない部分があることを鑑みメールクライアントは Roundcube を選択することとしました。後述しますが移管にあたつては他局から赤入れ機能などのサポートを要求されており、P

*5 SMTP サーバーについてはスパムメール発信マシンになつしまうリスクや、送つたメールがすべてスパムメール扱いされるリスクが、IMAP サーバについても落とすとすべてのメールが届かない大惨事リスクなどがあります。1~2 年で人が完全に入れ替わり、かつどのような人が入つてくれるかもわからない学園祭実行委員会のような学生組織においてそれを維持し続けるのは厳しすぎる

*6 実際にはスタンダードプランで契約した。最小構成の場合、メールボックス全体の容量としてはとくに文句がなかつたが、1 アカウントの最大容量が 1GB であり、かなり不安が大きかつた...というより窓口アドレスにおいては余裕で超えてしまうことが想像できた。スタンダードプランの場合には 20GB まで保存できる

*7 実委が通常送受信するようなメール（いわゆる HTML メールや日本語で書かれたメールなど）の送受信、閲覧が可能なこと、その他添付ファイルが処理可能であること、フォルダ機能があることなど

ラグイン開発ができることも要件の 1 つでした。

3 赤入れ

赤入れとは、書いた下書きに対して外部にメールを送りすぎて半ば専門家のようになった委員^{*8} がチェックを通してからメールを送信する仕組みです。いわゆる上長承認機能です。Google Workspace を利用していた時には、委員が下書きにメールを書き、赤入れ担当が下書きをチェックして書き直し^{*9}、連絡をして執筆者が送信するという流れを取っていました。

メールアドレスを委員が 1 つずつ持つようになると、下書きボックスという信頼できる唯一の情報源^{*10} がなくなってしまうので赤入れができなくなってしまいます。そこで、以下の流れになりました^{*11}。

- 執筆者が Roundcube のエディタから赤入れ依頼ボタンを押す
- プラグインがメールデータを Strapi に送信し、GitHub の Issue を立てる
- 赤入れ担当が Issue の文面を編集して、Issue を close する
- 執筆者がそれを反映させて送信する

とはいえる普通に毎日おおよそ 50 個ずつ Issue が立ち続けるの本当に意味不明なので、以下のような流れに直していく必要があります。

- 執筆者が Roundcube のエディタから赤入れ依頼ボタンを押す
- 赤入れ担当者が Roundcube から赤入れを行う
- 執筆者の下書き内容を自動的に書き換えて送信する

一方、それを実現するためには Roundcube のプラグイン内で謎のタグを駆使して新しいタブを実装する必要があります。ところでそのタグについての公式の説明がほとんどなく、サードパーティのプラグインを読んで祈りながら実験していますがまだ成功していません。

3.1 Roundcube プラグイン

Roundcube は PHP でスクリプトを書くことで独自の機能を追加できます。かなりプラグインの存在を前提にした設計になっており、たとえばアップロードされたファイルの管理や連絡先リストなど、メールクライアントとしてのコア機能もプラグインで実装されています。一方、プラグイン開発には以下のような障壁があります。

- 普通にドキュメントが全然更新されていない
- 渡ってくる引数が持つプロパティのリストがない
- UI 層を Roundcube が定義した謎のタグを使って書く必要がある

^{*8}赤入れを担当している友人はたまに知らない作法を教えてくれます。社会ってむずかしそう

^{*9}これ“発明”すぎる

^{*10}Gmailux

^{*11}マジで終わっている

1つ目はどうにもならず、2つ目は挙動を調べて独自のドキュメントを育てていくことでなんとかすることにします*12。

3つ目は Roundcube プラグインの責務をメールデータや赤入れサーバとの通信のみに抑えて、クライアントの処理は独自に書くことにしました。また、サーバ側がぐるぐるバージョンの揺らぎによって動作しなくなることがあり、開発時には Docker Compose を、ビルドには Nix を利用して*13*14 環境を整備しました。

```
1 $ exa
2 client
3 config
4 docker-compose.yml
5 docs
6 flake.lock
7 flake.nix
8 LICENSE-APACHE
9 LICENSE-MIT
10 Makefile
11 nix
12 README.md
13 scripts
14 server
15 strapi
```

簡単にソースコードを紹介します。Roundcube のプラグインは `rcube_plugin` クラスを継承して実装します。クライアント側ではプラグインから赤入れデータなどを受け取って、送信ボタンを削除したり赤入れ反映を行うような実装をしています。それらを `webpack` でバンドルしてサーバ側で読み込んでいます。また、赤入れ担当者が下書きの添付ファイルを確認できるように、逐次 `minio` サーバにアップロードしています。Roundcube はメールそれぞれに対して `unique` な ID を持つおらず、同じメールであってもセッションごとに ID が異なります。そこで、メールヘッダが持つ `messageID` が一意であると信じている*15ため、これを使って赤入れなどの処理を行っています。

*12 といったんね

*13 compose パッケージを Nix で固定できる。 <https://github.com/svanderburg/composer2nix>

*14 node パッケージを Nix で固定できる。 <https://github.com/svanderburg/node2nix>

*15 実際にその保証はなかったと記憶しています。Roundcube 側、あるいはさくらのメールサーバ側がこれを一意に設定していれば良いのですが、わかりません。

```
1 <?php
2     require_once __DIR__ . '/../../vendor/autoload.php';
3
4     class akairer extends rcube_plugin
5     {
6         use Attachments;
7         public $task = 'mail';
8         private $map;
9
10        function init()
11        {
12            $this->load_config();
13            $this->include_script('./dist/bundle.js');
14            // 下書きの添付ファイルの追加・削除状況をminioに反映する
15            $this->add_hook('attachment_upload', array($this, '
16                attachment_upload'));
17            $this->add_hook('attachment_delete', array($this, '
18                attachment_delete'));
19            $this->add_hook('message_ready', array($this, '
20                message_ready'));
21            $this->register_action('plugin.get_message_id', array(
22                $this, 'get_message_id'));
23            $this->register_action('plugin.send_to_mattermost',
24                array($this, 'send_to_mattermost'));
25            $this->register_action('plugin.test_strapi', array($this,
26                , 'test_strapi'));
27            $this->register_action('plugin.fetch_or_insert_email',
28                array($this, 'fetch_or_insert_email'));
29        }
30
31        function fetch_or_insert_email() { /* メールデータを
32            rStrapiに登録 */ }
33
34        function get_message_id() {
35            $rcmail = rcmail::get_instance();
36            $rcmail->output->command('plugin.get_message_id', array(
37                'message_id' => $_SESSION['message_id']));
38        }
39    }
40}
```

```

29     }
30
31     function message_ready($args)
32     {
33         $argsContents = print_r($args, true);
34         $rcmail = rcmail::get_instance();
35         $_SESSION['message_id'] = $args['message']->headers()['
36             Message-ID'];
37         $rcmail->output->command('plugin.message_ready_after',
38             array('message' => "Finshed"));
39         return $args;
40     }
41
42     function send_to_mattermost($message = NULL) { /*
43         Mattermostにログを送信 */
44 }

```

現在は手を回せていませんが、PHPStanなどの静的解析ツールを利用してできるだけ品質を維持できる状態で引き継ぎたいと思います。また、Strapiにラベルデータを管理して転送されたメールのmessageIDが同一であることを利用して、複数アドレス間で同じラベル情報を持つ機能も提供しようとしています^{*16}。

4 メールの受信・転送

さて、個人メールアドレスが生えたからといっていきなりすべての問い合わせ先として個人メールアドレスを晒すわけにはいきません。調べてみたところ、大体の法人等においては大まかな代表メールを設定し、そこに届いたメールを担当者に回覧させる方が一般的^{*17}ではあるようでした。そこで今まで用いていたinfo, project の2つのメールアドレスを代表メールとし、それらで受信したメールをすべて各実委人^{*18}に転送、各実委人はそれらに対して適時個人メールより返信を返すという方式を採用することとしました。しかし単純にメールを転送してしまうと添付ファイル10MBのメールが一通来るだけで10MB × 約300^{*19}=3GBを消費することとなり、学園祭実行委員会が契約しているさくらのレンタルサーバの全体容量であるところの300GBをあっという間に使い切ってしまいます。そこでいったん添付ファイルをいったん抜き、本文のみを転送するというスタイルを取ること |

^{*16}なら最初からメールクライアントを作った方が良かったのではないか？いや、それは見えない膨大な作業があるはずで。いやしかし...という会話を山ほどした。

^{*17}要出典。少なくとも筑波大学の事務方では割とやっているっぽい

^{*18}学園祭実行委員会（実委）に所属する人々のこと

^{*19}学園祭実行委員会に所属している委員の数。約300人。

にしました*20

4.1 実装

メール受信処理・転送スクリプトの実装としては概ね以下の流れを取ることとしました。

1. 各代表メールアドレスの受信トレイを 30 秒ごとに確認
2. もし新規メールがあった場合、メールをパース、以下の情報を取り出す
 - 送信元
 - 宛先
 - Cc 宛
 - 件名
 - 本文
 - 添付ファイル
3. 添付ファイルが存在する場合は Minio に上げる
4. パースして得た情報を Strapi に投げる
5. 送信元を偽装した上で同じ宛先・件名・本文のメールを作成、これに添付ファイルへのパスと Cc 情報をテキストとして最後に付け加え、各実委人に転送*21

メールの多様性

メールは多様な形式に対応するため当初の RFC5322 で定められたものからどんどん拡張を続けており、*22 メール形式にはさまざまなパターンが存在しています。主な変数としては以下のものがあります。

- よくある Content-Type
 - text/plain
 - text/html
 - multipart/alternative
 - multipart/mixed
- よくある Content-Transfer-Encoding
 - base64
 - quopri
- よくある日本語 Charset

*20もちろん筋筋スタイルを採用し、容量を消費するたびにアップグレードするというのもひとつの手ではあるが、ファイルサイズが 300 倍されてくる状態では焼け石に水どころではないしそんな予算があったら Google Workspace を人数分契約している

*21この流れを構想した際は 1 日で終わると思っていた実装、まさかこれが地獄の始まりだとは思いもしていなかった.....

*22ここ、非 ASCII 想定していないから後出し RFC で結局“多様性”が発生してるしなんか charset 統一されていなかつてか添付ファイルはどんなにでかくても base64 としてメッセージ内に普通に埋め込まれているとか辛さがそこそこあったので書きたかったけどなんと締め切り超過しているので断念.....

- ISO-2022JP
- UTF-8

これらの変数についてはメールクライアントによってどれを採用するかがまちまちであるものの、通常 Content-Type や Content-Transfer-Encoding、Charset としてどれを採用しているかについてはメッセージ内にて宣言されることとなっており、楽勝～と思ひながら実装に進みました。そしてそれを本番運用に投入したところびっくり仰天、実委に届くメールの肌感 10~20% がこれらの変数をまったく宣言していないか、ひどい場合だと宣言しているものの実際が異なるという手に負えないものすら存在しました。こうした結果これらの変数を順番にすべてトライし、一番正常っぽい結果を出したものを転送するという筋筋実装を行うこととなっていました。^{*23}

5 アカウントの紐付け

さくらのメールボックスは CSV でまとめてメールアドレスを発行するといった機能がありません^{*24} ので、何もしなければ手動で 300 ものメールアドレスを頑張って作る必要があります。これはあまりにも苦痛すぎるので、アカウントを自動発行できるような仕組みを整える必要があります。

まず、大学が発行するメールアドレスである、@u.tsukuba.ac.jp を利用することが考えられます。しかし、学生の中には学実委ではない方も含まれていることが明らかになつたため、この方法では実委人のみを絞り込むことは難しいです。

学実委では作業関連のコミュニケーションのために Slack のワークスペースを利用しているのですが、これに参加している人は基本的に実委人であると見なされています。そこで今回は、ワークスペースのユーザ ID を持っていることを実委人であるという証明にして、以下の機能を満たすソフトウェアを実装しました。

- Google Form を使って必要な情報を収集する
- さくらのメールボックスでメールアドレスを作る
 - rust-headless-chrome を使って自動作成されるようにした^{*25}
- そのメールアドレスで Auth0 のアカウントを作る
 - API を用意してくれてありがとう
- それらのパスワードは Bitwarden CLI で生成して Send リンクを作成し、Slack の DM に送信する
 - API を用意してくれてありがとう

*23 メールクライアントによっては返信を連ねた際、個々のメールを multipart 内のブロックとして分割するのですが、それぞれのメッセージを送信する際に違うメールクライアントを使っていると同じプレインテキスト チックでも Content-Type がずれまくったり、個々で Charset が違う……！？ みたいな謎事象が発生したりと謎で辛かった。

*24 さくらのレンタルサーバ、期待しながら探したんだけど普通に API が存在しなかった 残念……

*25 call_js_fn というメソッドに JavaScript コードである文字列を渡すとそのまま実行してくれる謎機能があつてウケていたらそれを結局使うことになってしまった

これもありまともではないのですが、時間と気力が全然ありませんでした。せめて Slack の DM で完結していると良いと思う。

Auth0 を導入したのはかなり良くて、情報メディアシステム局が実委人に向けて内部向けに提供しているサービスに SSO ができるようになりました。Roundcube にも導入できると良いですが、契約しているさくらのメールボックスのプランでは SSO は提供されておらず、セキュリティ周りの知識がなかったため安全に Auth0 を介してログインさせる方法がわかりませんでした。後で期待

6 現状の課題

現状のメールシステムには以下のような課題があります。

- 送信したメールが他者によって確認しづらい
 - 基本的に実委人サイドからは個人アドレスよりメールを送付するため、他者が送信内容や、とくに送信ステータスを確認するには本人に直接聞くか、GitHub の Issues を漁る他なくなってしまう
- 予約送信が出来ない
 - メール界隈では相手が活動している時間にメールを送信することがマナーであるらしい²⁶。実委人は夜に生きているが、一般的な社会は昼に動いているため、そのギャップを埋めるために予約投稿が必要
- スパムメールの検知・隔離
 - さくらのメールはスパムメールの検出方法として SpamAssassin を用いているが、これは以前使っていた Gmail の迷惑メールフィルタに比べるとどうしても検知精度がある程度落ちてしまい、結果迷惑メールが受信トレイに紛れ込むことが多くなってしまっている
- Roundcube の検索が貧弱
 - 過去のメールはバックアップ用のアドレスから検索するが、Roundcube の検索は 1 万件を超えると非常に重くなり検索結果を返せないことも多々ある
- 個人宛に送信されたメールの行方を追うことができない
 - Google アカウントが共有されていた頃は、良くも悪くもすべてのメール追うことができたため、責任者放置されているメールを巻き取れたが、現状では個人宛にのみ送信されたメールの進捗状態が確認できない。しかし個人宛の 2 段階認証メールなどのみを上手く弾き出す方法が思いつかなかった。

システム的にはかなり不十分なのですが、Roundcube プラグインの開発が後で以降も続くことで、長期的に見れば下書きをチェックするよりはまともになるかな²⁷ と思っていた

*26 なんでだよ

*27 下書きボックスを使って赤入れをするのは、執筆者がリロードしないままメールを送信してしまうと、赤入れによって書き換えた文面は消滅して元のメール文で送信されてしまうとか、そもそも赤入れを通さずにメールを出してしまった人がいるとか、それはそれで大変な部分もあったようです。システムがそれを握ることは結構

節があります。後代に期待(2)

一方で、2年で^{*28} 人員が入れ替わる環境で強行するべきではなかったのかなども思っています。

7 むすびに

確かに 400 万円も請求したくなる

大切なと思うのですが、全然時間がなくて破綻箇所がたくさんあり、かなり申し訳ない……

*28 これは学実委としては 2 年弱所属できるが、私は昨年度は先輩の仕事を見ながらすごいなあと思ってほんやりしていたので実質 1 年弱。1 年生から動ける人は動いた方が理解度が上がってより楽しめるかもしれないです。

Openfab 創房を知っておこう

文 編集部 てばさき (@tebaskit)

1 はじめに

Openfab 創房^{*1*2}(以下 創房) の 2023 年度のリーダーを務めているてばさきです。創房はキャンパス OJT 型産学連携教育推進財団(以下 COJT 財団)と情報理工学位プログラムが共同で開設・運用している機能複合型多目的実習室です。創房の使い方がわからない・そもそも知らないという人が多いと感じたので、本記事では創房の概要と施設の紹介、そして創房の使い方について説明します。

2 創房概要

創房は粉クリや食堂がある 3C 棟隣の工学系 E 棟 203 にあります。次章で説明する 3D プリンターやレーザーカッターをはじめとする様々な加工機を設置しています。学生インストラクターが開設している WelcomeHour では学群学類、目的問わず学生が自由に創房を利用できます(週に 2,3 日程度しかやっておらずすまぬ)。また、COJT(組み込みキャンパス OJT)^{*3} や情報特別演習といった授業でも創房を利用することができます。

創房の考古学・以前の様子については過去記事^{*4} を参照ください。

3 創房施設紹介

実際の作例を見せられませんが、創房に設置されている様々な加工機を紹介します。

3.1 3D プリンター

創房には熱積層方式と光造形方式の 2 種類の 3D プリンターを設置しています(図 2)。

熱積層方式 3D プリンター

熱積層方式の 3D プリンターは 3 機種あり、それぞれ異なる特徴を持っています。

図 2a は Flashforge Adventure 3 です。特に要件がないのなら、こちらの 3D プリンターで大体印刷できます。また、こちらの 3D プリンターならどのインストラクターでも対応できます。創房のフィラメントの在庫的に材質は PLA で、色は白や黒を使うことが多いです。

図 2b は Creality Cerman D1 です。こちらの 3D プリンターは創房の中で大きいものを印刷できます(280 x 260 x 310 mm, Flsahforge の Adventure3 150 x 150 x 150mm)。PLA の

*1 https://twitter.com/openfab_sobo

*2 http://www.cojt.or.jp/tkb/openfab_sobo/index.html

*3 COJT 財団が主宰する授業で、主に情報学群の 3 年次を対象にしている。<http://www.cojt.or.jp/tkb/index.html>

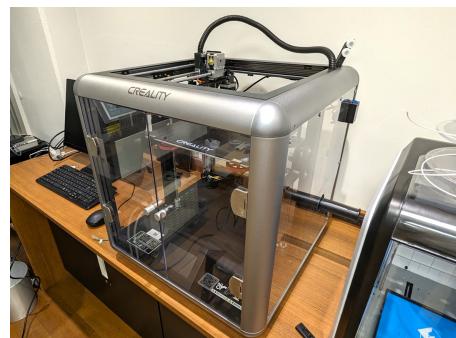
*4 WORD38 号 <https://www.word-ac.net/post/2016/0308/> : 「openfab 創房」前編: 室内概観・機材類紹介



図 1: 創房の全体画像



(a) Flsahforge Adventure 3



(b) Creality Cermon D1



(c) Qholia



(d) 光造形方式の 3D プリンター

図 2: 創房の 3D プリンター達

フィラメントだけでなく、PloyFlex^{*5} といった特殊なフィラメントも印刷できます。ただし、こちらの 3D プリンターはインストラクターが対応している人が限られています。また、印刷品質が悪いため、大きさが必要ないのなら Flashforge Adventure 3 を使うことをおすすめします。

図 2c は今年にオーバーホール(修理)されて、再び使えるようになった 3D プリンターです。他の 3D プリンターよりノズルが細いことから、高い印刷品質を誇ります。ただし、こちらの 3D プリンターはインストラクターが対応している人がいないことから、現状は技術室の職員^{*6}に対応してもらう必要があります。

光造形式 3D プリンター

図 2d は光造形方式の 3D プリンターです。光造形方式の 3D プリンターは熱積層方式の 3D プリンターとは異なり、紫外線硬化樹脂を用いて印刷します。創房では水洗いレジンは洗浄処理が簡単な水洗いレジンを標準で使用しています。熱積層方式より高い解像度で印刷できます。現状は技術室の職員に対応してもらう必要があります。

3.2 レーザーカッター



図 3: レーザーカッター

*5 <https://www.poly-maker.jp/polyflex.html>

*6 工学系 E 棟 101 システム情報技術室 創房担当 米田さん

図 3 のレーザーカッターでアクリル板や木材、紙などを切ることができます。Adobe Illustrator や CAD によって形状を決めることができます。後述の UV プリンターと一緒に使うことでアクリルフィギュアやアクリルキーホルダーなどを作成する事例がよく見られます。

3.3 UV プリンター



図 4: UV プリンター

図 4 は紙だけでなく、アクリル板や金属版にも印刷が可能な UV プリンターです。対応できるインストラクターが限られているため、事前に連絡をくださると助かります。

3.4 CNC マシン

図 5 の CNC マシンで木材やアクリル板、金属板を切削することができます。銅板を掘削することで、電子回路基板を作成する例もあります。こちらの加工機は対応できるインストラクターが限られています。

3.5 吹付機

図 6 の吹付機で金属板に塗装することができます。使ったところを見たことないため、詳しいことはわかりません。使いたい方は、技術室に聞いてみてください。

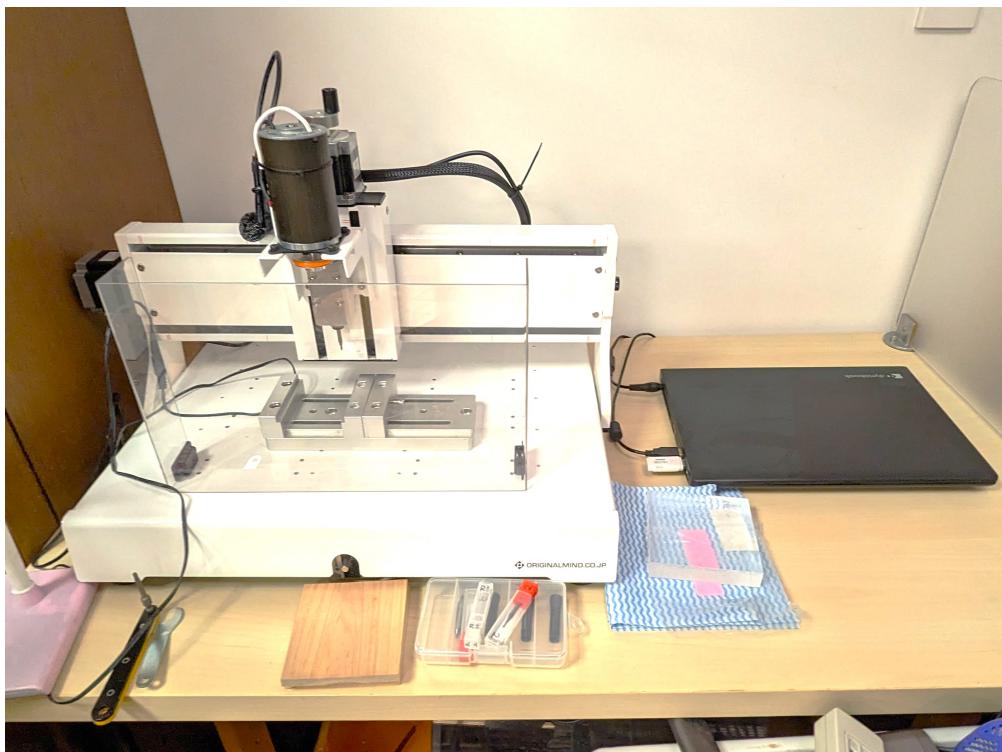


図 5: CNC マシン

3.6 ハンダゴテ、吸引機、その他

最後に、3D プリンターでの印刷やレーザーカットした後に細かい加工を行いたい場合に必要となるヤスリやニッパなども用意してあります。また、電子工作で必要なハンダゴテや吸引機も用意してあります。

4 創房の使い方

創房に加工を依頼する方法はオンライン委託加工と対面委託加工の 2 種類があります。

4.1 オンライン委託加工

オンライン委託加工は創房 Twitter のトップツイート^{*7} に記載されている、フォームから加工データを入力することで加工を依頼することができます。

加工が終了すると創房からメールが届き、ロッカーや対面での受け取り、支払いができます。

*7 https://twitter.com/openfab_sobo/status/1351787613967925248

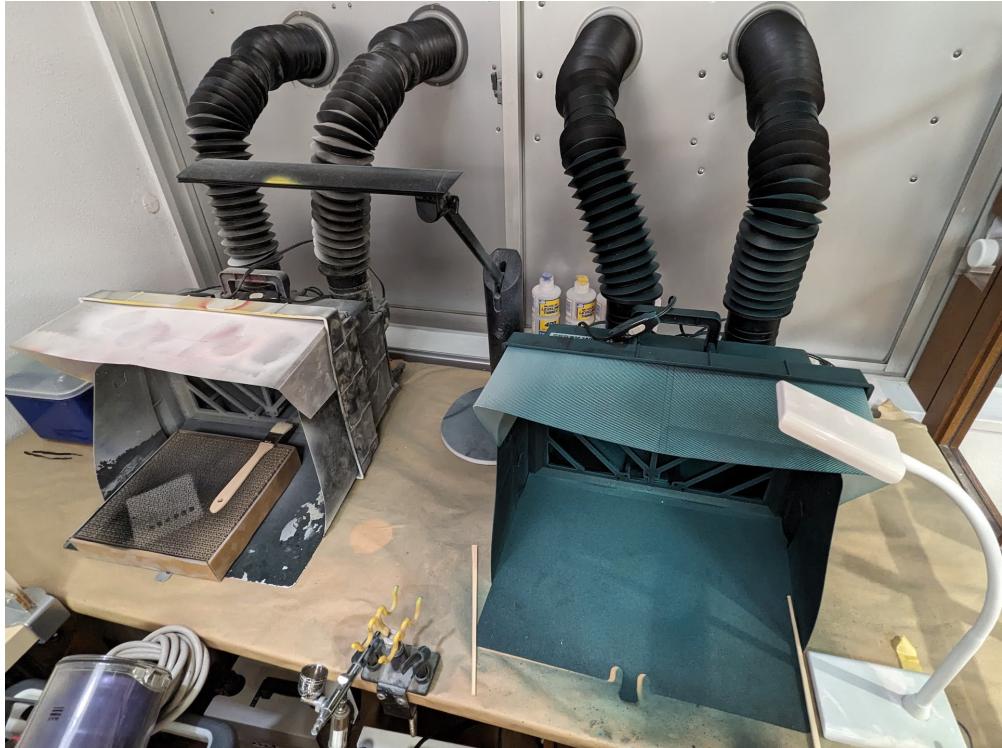


図 6: 吹付機

4.2 対面委託加工

対面委託加工は創房に直接来て加工を依頼する方法です。WelcomeHour の開設日は毎週日曜日に創房 Twitter で告知(図 7)しています。

 **openfab創房@筑波大3E203 / C102 日曜西え39b** ...
@openfab_sobo

【WelcomeHour開催のお知らせ】
下記の日程で実施します。創房 工学系E棟203に是非お越しください [17]
8/2 10:00~12:00
8/3 18:00~20:00
8/4 18:00~20:00

午後3:00・2023年7月31日・264 件の表示

図 7: 創房告知ツイートの例

創房に来たら、まずはインストラクターに加工を依頼する旨を伝えます。希望の印刷機や加工方法、データをインストラクターに渡します。加工終了後に指定された料金を支払

い、加工を終えます。対応しているインストラクターによっては加工を行えない場合があります。

5 おわりに

質問があれば創房の Twitter やメール^{*8}で問い合わせてください。

^{*8}customer_openfab_sobo@googlegroups.com

Linux で絵を描いてみよう

文 編集部 高尾創介 (omemoji)

1 はじめに



図 1: Linux 上で Krita を使用して描いた絵。アステカ神話の「鷺の戦士」をモデルとしている

意外に思われるかも知れないが、Linux を使って絵を描くことは難しくない。筆者は Linux を普段使いしており、Linux で Krita (クロスプラットフォームのペイントソフト。後述) を用いて絵を描いている。

本記事では Linux で絵を描く際の環境構築、並びに実際に絵を描いてみての所感を記述

する。

- Linux を普段使いしているが、どうやって絵を描いたらいいか分からぬ
- ペイントソフトに金を払わず安価でお絵描き環境を構築したい
- CLIP STUDIO PAINT が使えない OS でどうやって絵をお書きになるのですか?

という風に考えている方は是非本記事を読み、Linux での創作活動を楽しんで頂きたい。

2 環境構築

2.1 PC とペンタブレット

まず、必要なデバイスについて説明する。

1. Linux を入れる PC
2. ペンタブレット

の 2 つがあれば、絵を描くには十分である。以下、それについて説明する。

Linux を入れる PC

ディストリビューション選びやインストールの方法については割愛するので、各自調べて欲しい。

PC の性能はそこまで高くなくてもよく、ラップトップで十分快適に作業を行うことが出来る。目安として、筆者が普段使っている PC のスペックや価格を以下に示す:

機種: Lenovo ThinkPad E14 Gen 3 (AMD)

OS: Void Linux

CPU: Ryzen 5 5500U

GPU: Radeon Graphics

メモリ: 12GB *1

価格: 約 6.5 万円 *2

注意することとして、PC の液晶に TN 液晶を選ぶべきではない*3。多少価格が高くなつても、IPS 液晶のものを選択するべきである。

ペンタブレット

ペンタブレットは絵を描く際に用いる入力機器であり、大きく

- 板タブレット (以下、板タブ)
- 液晶タブレット (以下、液タブ)

*1 18GB でも問題なく作業出来る。筆者はメモリを増設して 16GB にした

*2 メモリの増設や液晶の換装などを行ったため、最終的に 8.5 万円ほど費やした

*3 視野角が狭く、少し角度をつけて画面を見ただけでも見える色が大きく変わってしまう

の 2 つに区分される。この 2 つを比較した際、板タブのメリットとしては

- 安い
- 持ち運びやすく外での作業に適している
- 比較的場所をとらない

などがある。一方、液タブは板タブと比較して高価で嵩張るもの、

- 手元を見ながらアナログに近い感覚で絵を描ける

という非常に大きなメリットがある。

それぞれの長所を踏まえた上で、あくまで本記事においては「価格や持ち運びやすさに優れ、気軽に導入出来る」という点で板タブをおすすめしたい^{*4}。また、大体の Linux ディストリビューションで Wacom デバイス用のドライバが標準搭載されているため、メーカーにこだわりがなければ Wacom 製のタブレットを購入することを勧める。

ペンタブレットの入手については、メルカリを始めとしたフリマアプリを使うと安価で購入することが出来る。「ほぼ新品」「数回使っただけ」の商品がよく出回っている^{*5}ため、品質についてあまり心配する必要はない。例えば、筆者はメルカリでほぼ新品の板タブ (Wacom Intuos Medium CTL-6100) を 8,450 円^{*6}で購入することが出来た。

2.2 ソフトウェア

次に、絵を描くためのソフトウェアについて説明する。Linux では、残念ながら

- CLIP STUDIO PAINT
- Procreate
- ibisPaint
- MediBang Paint

といった有名なペイントソフトは使用出来ない。その代わりとして、本記事では「Krita」というペイントソフトを紹介する。

Krita

Krita はオープンソースのペイントソフトであり、Linux だけではなく Windows や macOS でも動作するクロスプラットフォームソフトウェアである。上記したような有名なペイントソフトと異なり完全に無料でありながら、それらと遜色ない程豊富な機能を持つ。

インストールに関しては、apt や dnf など多くのパッケージマネージャに対応している。また、flatpak でインストールすることも可能である。

使い方については、UI やショートカットが独特であるため多少の慣れが必要である。簡

*4持ち運びやすいため、授業の板書にも使うことが出来る

*5恐らくすぐに挫折したのだろう

*6公式ストアでは 17,380 円もする (<https://estore.wacom.jp/ja-JP/products/wacom-intuos/ctl-6100-k0.html>, 2023 年 10 月 22 日閲覧)

単な使い方は <https://esinote.com/guide/knowledge/krita> を参考にするとよい。

3 実際に描いてみて

高度なことをしていないだけかも知れないが、今のところ特に不自由を感じることもなく作業出来ている。Krita については特にデフォルトのブラシが優秀で、鉛筆、エアブラシ、水彩や G ペンなど自分が欲しいブラシは大抵存在した。また、カラーバランスの調整やフィルターなどの標準的な画像編集機能がある点も評価出来る。それなりのラップトップと板タブに無料ソフトを入れるだけで手に入る環境としては十分過ぎるほどであり、今後の作品制作にも十分なパフォーマンスを発揮すると期待出来る^{*7}。

ただし、Linux を普段使いした上で Krita で絵を描く場合、

- 標準的な環境ではないため、特に商業レベルの制作ではサポートされにくい
- 文献の多くが英語であるため、慣れていないと大変
- 企業のサポートが受けられないため、トラブルは自分で調べて解決しなければならない

といった苦労もある。ユーザーコミュニティや親切なサポートを含めた総合的な使いやすさを重視する場合は、CLIP STUDIO PAINT を始めとした人気ペイントソフトを使う方がよいだろう。

4 最後に

Krita に関してもっとよく知りたい場合は、ドキュメント (<https://docs.krita.org/ja/>) を参照のこと。

また、筆者がこれまでに Krita などで描いた作品はポートフォリオサイト「創作物紹介」(<https://omememoji.com>) で公開している。

^{*7} 尤も、絵を描く上で最も重要なのは道具ではなくそれを扱う人間の技術である

原動機付自転車はいいぞ

文 編集部 onokatio

1 つくば市の道路には原付が向いている理由

筑波大生の多くは自転車を所有している。筑波大学もといつくば市は南北に長く、徒歩での移動は困難だからだ。しかしつくば市の道路は街路樹の根や老朽化したアスファルトにより凹凸が激しく、例えばママチャリやクロスバイクで走行すると振動に悩まされることになる。マウンテンバイクを買うのも一つの手であるが、本節では原動機付自転車（以下、原付）を推奨する。

2 乗り心地について

まず第一に原付は車道を走行するため、歩道の凹凸に悩まされることはない。つくば市の道路は、車道の方がある程度路面の状態がいいと言える。歩道を走行している時に気にする段差や障害物は、車道を走行する時には気にする必要がない。次に最も重要な点として、原付にはサスペンションがついている。これにより、路面の凹凸による振動を吸収することができる。自転車であればマウンテンバイクにしか搭載されていないが、原付は基本的な車種には搭載されている。その他の点としては、人力を動力とする自転車と違って路面抵抗をそれほど気にする必要がないため、タイヤを太く大きくすることができる。タイヤの空気量が大きい・圧力が小さいほどタイヤに力が加わった際の変形が大きくなるため、タイヤだけでもある程度の振動を吸収することができる。クロスバイクの細いタイヤを利用した経験があればこれはすぐに理解できるだろう。

3 費用について

原付の費用は安い。これも原付を推奨する理由の一つである。原付を使うためには主に3つの費用がかかる。

- 免許取得費
- 購入費
- 維持費・燃料費

3.1 免許取得費

免許取得費は、原付を運転するための免許を取得する費用である。普通自動車免許を所持している場合は内包されているため新規に取る必要はないが、所持していない場合は新規に取る必要がある。原付免許は普通自動車免許と異なり実技試験が存在しない。学科試験のみである。ただ注意点として試験合格後に「講習」という名の実技の授業を受ける必

要がある。2023年現在での茨城県警での試験料は3550円^{*1}である。実技講習は、大抵の自動車教習所に行けば受けることができ、料金は4千～5千円程度である。免許取得費は、これらの費用の合計である。

3.2 購入費

購入費は原付を買うときに必要となる料金である。スクーター・ビジネスバイクは高くても中古では10万円程度で手に入れることができる。筆者はつくばローカルコミュニティ^{*2}で6万円でスーパーカブを購入した。ナンバープレートの発行などはお店側が行ってくれることが多く、追加で料金が発生することは少ない。

3.3 維持費・燃料費

維持費は主にエンジンオイルの費用、税金、保険の三つである。自動車と違って車検は存在せず、日々の点検はエンジンオイルを交換するだけで良い。もちろん故障や事故などを起こせばその度に費用がかかるが、長期的に見積っても無視できる確率と言える。エンジンオイルは半年に一回程度交換する必要があり、大抵の原付は1L 1000円程度のオイル代金がかかる。業者に依頼すれば工賃がかかるが自分で交換することもでき、特別な工具なども必要ないため自力をお勧めする。（ここでは具体的な方法は割愛する）税金に関しては、原動機付自転車(50cc以下)の自動車税は年額2000円^{*3}であり、安価である。これはつくば市の金額であるが、全国どこの自治体でも変わりはない。

保険については、強制保険（自賠責保険）と任意保険が存在する。強制保険は原付を購入する際に自動的に加入される保険であり、任意保険は自分で加入する保険である。強制保険は年間6910円であるが、5年間の保険料を一括で支払うと割引が適用され一年あたり2662円になる。途中脱退する場合は返金もあるため、先納を前提とすると安価と言える。任意保険については自分で選ぶことができるため、保険料は様々であるが、原付のみの保険に加入する必要はなくほとんどの場合自動車保険の付帯で十分である。これはファミリーバイク特約と呼ばれ、現在加入している自動車保険に追加で料金を支払うことで、保険契約主とその家族がバイクに乗った際の保険を付帯するものである。筆者は両親の自動車保険に付帯しているファミリーバイク特約を利用しているため、年額5000円程度で対人対物無制限である。

燃料費はガソリンの費用である。年々石油が高騰しているが、原付は本体が軽量という点から燃費が良い。一般的なスクーターは70km/L～120km/L程度であると言われており、これは自動車が10km/L～50km/L程度であることを考えると非常に燃費が良いと言える。ガソリンは1Lが130～170円程度であるため、100km/Lと仮定すると1km走行するために1.3～1.7円の費用がかかる。

^{*1}https://www.pref.ibaraki.jp/kenkei/a03_license/exam/gentsuki.html

^{*2}<http://tsukuba-lc.com/>

^{*3}<https://www.city.tsukuba.lg.jp/soshikikarasagasu/zaimubushiminzeika/gyomuannai/4/1/1001043.html>

4 まとめ

以上で説明したように、自転車と比べて費用は発生するが、学生にとっても出せない額ではなく、乗り心地や移動時間の短縮、移動範囲の広がりを考えるとメリットのほうが大きい。また原付で全ての移動を賄う必要はなく、例えば自転車やカーシェアと併用する手もある。これから自転車を購入しようと悩んでいる方、また既に自転車を所有している方も、一度検討してみてはどうだろうか。

オタク京都旅行記

文 編集部 yuseiito

1 何

ノートの取り方について、それなりに分厚くて人によっては有用な記事を書こうと思って内容の整理だけして寝かせていたら、気づいたら（自分が設定した）記事締め切りの1時間前でびっくりした^{*1} yuseiito が、夏休みに coins21 の友人たちと京都に遊びに行った時を振り返って、薄くてあまり有用でない記事を書くことにしました。

2 1日目

一日目は、移動でした。つくばでアイシスを借りて、オタク3人が交代しながら京都まで運転していました。私は、運転せずに^{*2}クリーンアーキテクチャをどこまで真面目に実践するかについて別のオタクと語っていました。

10時につくばを出発して、途中沼津でかの有名なハンバーグ屋さわやかでランチを食べました。うまかった。



図1: さわやかのハンバーグ

その後、また休憩を挟みながら新東名・新名神を走って大津SAへ。名古屋あたりで強烈な雨が降ってきて、運転していた編集部員でもある maetinくんが大変そうでした。

私は横で「名古屋港! 紹麗!」とさわいでいました。助手席に向いてない。

時間も遅かったので、大津SAで夕食に。私はフードコートの肉うどんを食べました。

大津SAには琵琶湖が見える展望デッキがあるので、そこで琵琶湖の夜景も見ました。実

^{*1}コードも日本語も、どうすればできるかがわかった瞬間に急激にやる気が削がれるのはなんなんだろう。

^{*2}そういうえば、この夏休みに免許を取った。そのことについて書いた方が本当は有用だったと思う。



図 2: 肉うどん

は私は滋賀県民で、琵琶湖に強い愛があるので、そこでもオタク特有の早口をやりました。恥ずかしい。

それはそうと、琵琶湖の夜景はいいですよ。名神を走られる際にはぜひ大津 SA へ。

その後、また走り出し、夜（実のところ正確な時刻をもう忘れた）、京都に到着して、宿にチェックイン。宿は、Airbnb でとっておいたコンドミニアムで、5 人部屋。その日はもう遅かったので、おやつを食べてダラダラと語ったのち就寝しました。



図 3: 関西な感じのお菓子たち。

ダブルベッド 2 つ + ソファベッドの構成の部屋だったので、オタクとダブルベッドで寝るというなんともアレな経験をしました。一人、酒をよく飲むオタクだけが一人外に飲みに行っていました。かなり当たりの店が見つかったようで、嬉しそうでほっこりしました。

3 2日目

二日目は、どベタな京都観光をしました。というのも、実のところこの旅行自体がピザと一緒に食っていた時に「京都に行きたい！」と突然なったオタクの一言から唐突に計画されたもので、京都のどこに行くかは全く決まっていなかったのです。

そこで、ひとまず伏見稻荷でも行くかということになり、コンビニで^{*3}朝食を買って車内で食べてから伏見稻荷へ。



図 4: 伏見稻荷

そこでの長い階段を全て上りましたということになると extreme でかなりオタク旅行っぽいのですが、普通に体力も気力も時間もないということで、早々に引き返しました。外国人が多かったのが印象的でした。

その後、嵐山でも行くかということになり、渡月橋近くの駐車場に車を止めました。そこから渡月橋をめざす途中に、「電気電波関係業界の発展を祈願するため」^{*4} の法輪寺の社殿である「電電宮」があるねということになり、全員 coins なので行っておきました。

スポンサー^{*5} の掲示をみて、就活はここに掲示のある企業に順番に応募すると良いのではと思いましたが、あまりいい噂を聞かないある SIer の名前があるのを見て萎えました。

その後、渡月橋をわたり、桂川沿いに歩いてお店を回りました。いい感じのお抹茶が買えたのが嬉しかったです。そこでランチを済ませ、次なる目的地は……どこにしよう。

ヤバいです、ネタ切れです。何も考えずに京都に來たので、どこに行けばいいのかよく決まっていないのです。もちろん有名どころの観光地はいろいろあるのですが、行ったことがあるところも多いので、いまひとつピンときません。

結局、猫の好きなオタクが「^{にゃんにゃんじ} 猫猫寺」に行きたいというので、行きました。これは誤解を生みやすい名称なのですが、猫猫寺は猫がいる寺ではありません。

猫もいないし、寺でもないのです。ただ、猫の置物や猫をモチーフにしたアート類が多く

^{*3}なんか京都らしいもん食えよ。

^{*4}<https://www.kokuzohourinji.com/dendengu.html>

^{*5}寺社に金を払った人や会社のことを表す語彙、ほんまにこれで合ってんのか？



図 5: 電電宮



図 6: 猫猫寺

展示されている「テーマパーク」だそうです。^{*6}なかなかかけたいなところだなあと思いましたが、ああいう機会でないと行かないので良かったと思います。

猫猫寺を後にして、まだ時間があったので京都文化博物館を眺めてから宿へ戻りました。そこから夕食へ。なかなか良い感じの居酒屋で、串かつやら各種の小鉢やらをいただきました。かなり幸せで、オタクたちと「人生の答え合わせやね」と言い合っていました。

その後、カラオケに行きました。これは、実のところ私が「カラオケに行った事がない」「カラオケ概論の実施が待たれる」という趣旨の発言をしたことがきっかけで、私にとって人生初のカラオケでした。だからどうということもないですが、音が大きいなと思いました。

帰って、寝ました。

*6 モチーフとなっている猫も、リアルな猫というよりはかなり強くデフォルメされた猫である傾向を感じた。

*7 私は飲酒も喫煙もしないので、かなり不思議に思っている。とはいっても、こんなヘンな写真の掲載を許してくれた maetinくんには感謝するばかりである。彼の記事「バカとバスこそインターネットに行け！」はもっと有益なのでぜひお読みいただきたい。

図 7: ヤニと酒で最高になる maetinくん^{*7}

図 8: メシの様子

4 3日目

移動しました。よく考えると、3日の旅行のうち2日は移動していることになります。朝食を大津SAで食べました。^{*8} 私はブラックバス天丼を食べました。ブラックバスは、琵琶湖でその被害が深刻となっている外来種で、釣り人はリリースせずに湖岸の回収ボックスに入れることで、個体数を減らす取り組みがあります。



図 9: ブラックバス天丼

ブラックバス天丼は、そのブラックバスの天丼です。特に臭みもなく、美味しかったです。その後は浜名湖までひたすら走り、浜名湖SAで休憩しました。浜名湖SAでは、桜えびのラーメンと浜松餃子を食べました。^{*9}

その後また挟みながら東名を走って帰ってきました。海は綺麗でしたが、富士山は見えませんでした。

圈央道を走って、8時ごろにはつくばに戻ってきました。

^{*8}本当は551が食べたかったのだが、朝早かったのでまだ開いていなかった。

^{*9}やけにメシの話が多いな。



図 10: 桜えびのラーメン

5まとめ

全体として、非常に楽しめた旅行でした。^{*10}

全体としてテックと人生の話が多かったですが、それはいつものことで良かったと思います。

と、ここまでつらつらと書きましたが、本当に有用でもなんでもない記事ができてしまいました。

本来 wordian としてはもっとけったいな旅行をしたり、おかしなところを訪問したりすべきところで、少し残念に思われた方もいらっしゃるかもしれません、たまにはメシの話だけでもいいのではと思います。

というわけで、これ以上学類の紙を無駄にするのは遠慮して、この記事はここで終わりにしたいと思います。coins 生の普通の夏休みの思い出の一コマでした。^{*11}

*10主に移動でしたが。

*11受験生や先生方には逆に珍しかったりしない……? しないか、そうだよね……

宅配物の状況を CLI で一元管理する

文 編集部 lapla

1 はじめに

coins21 の lapla といいます。2日前（2023/10/20 14:20 JST頃）にWORD 編集部に入り、現在（2023/10/22 3:30 JST頃）初記事を書けという圧に押されてキーボードを叩いています。

従って何を書こうか考えあぐねていたわけですが、ベタに最近作ったものの話をしようと思ったため、「宅配物をコマンドライン上で一元管理できる CLI アプリケーションを作った話」について書きます。このアプリケーションのコードは <https://github.com/lapla-cogito/takuhai> にあります。

2 作成の動機

特にオンライン上での購買などの行動を行うと、購入した物品が物理的媒体を伴う場合、その多くは宅配物として発送され、注文者の元に届きます。そして、多くの宅配事業者はこの宅配物に追跡番号と呼ばれる番号を振り、顧客が宅配物の詳細な動きを確認できるようにしています。

しかしこれには大きく次のような問題点があると考えました：

- ・現在自分の元に来ていない宅配物の一覧を簡単に確認できない
- ・追跡番号はメールで来る場合が多いが、これは雑多なメールに埋もれる可能性がある
- ・各宅配事業者の追跡番号検索ページに行き、番号をコピペする手間が生じる^{*1}
- ・追跡番号だけでは、どのような荷物かを推測するのが困難

このため、コマンドライン上でコマンドを叩くだけでこれらの問題を解決できるようなアプリケーションを作つてみました。

3 簡単な使い方

使用法の詳細はリポジトリの Readme に記載してあるため、ここでは主要な機能の紹介に絞ります。

3.1 宅配物を登録する

例えば、佐川急便が配送を担当する、追跡番号が 1234567890 の荷物があったとします。この宅配物に「荷物」というエイリアスを付けて登録したい場合、コマンドライン上で以

^{*1}これは例えば1つの注文が複数の発送に分かれて、追跡番号が個別のメールで通知されると特に面倒だと思います（実際このようなケースには遭遇したことがある）

下のコマンドを叩きます：

```
$ takuhai reg --sagawa --tracknum 1234567890 --name 荷物
```

このコマンドにより、裏で対応する宅配事業者（ここでは佐川急便）のページに HTTP リクエストが飛び、レスポンスをパースすることで、「〇〇時 ×× 分に□□営業所に到着した」等の情報を取得し、ローカルに保存します。

3.2 登録した宅配物の状況を追跡する

先ほど登録した、「荷物」というエイリアスを付けた宅配物の状況を追跡したい場合、コマンドライン上で以下のコマンドを叩きます：

```
$ takuhai show --name 荷物
```

この際も、対応する宅配物の情報を得る HTTP リクエストを飛ばして最新の宅配物の状況を取得します。

3.3 その他の機能

例えばエイリアスの付け替えや、登録した宅配物をローカルから削除する機能、登録した宅配物のいくつかを選んでエクスポートしたり、他人がエクスポートしたものインポートする機能を有しています。

このエクスポート・インポート機能についてもう少し説明すると、世の中には、複数人宛てであるが、宛て先はどこか 1 か所であるという例^{*2} が存在します。このような場合、該当する荷物だけを指定してエクスポートし、他の人にインポートしてもらうことで、宅配物の状況を各人がコマンドライン上で追跡できるようになります。

4 技術スタック

4.1 cobra

本アプリケーションは Go で実装しました。Go はほとんど触れたことがありませんでしたが、ササッと CLI アプリケーションを作る場合、cobra^{*3} というライブラリが有用であるそうで、実際に k8s や Docker での CLI の実装にも用いられているようなので、触れたくなったというのが理由です。

cobra は、cobra-cli コマンドを go install することで大変便利に開発を行えます。例えば、\$ cobra-cli init とすることで、cobra を用いたアプリケーションに必要な、次のファイル群が生成されます：

^{*2}筆者は現在ある会社を知り合いと経営していますが、全員（4 人）分の名刺を発注する際に全員分の名刺がまとめて CEO 宅に届けられたことがあります

^{*3}<https://github.com/spf13/cobra>

```

1 $ cd sample
2 $ tree .
3 .
4 |--- LICENSE
5 |--- cmd
6 |   |--- root.go
7 |--- go.mod
8 |--- go.sum
9 |--- main.go
10
11 1 directory, 5 files

```

また, `$ cobra-cli add hoge` とすることで, `cmd/hoge.go` が生成され, サブコマンド `hoge` を追加することができます。ここで生成した `root.go` や `hoge.go` の中身では, `cobra.Command` 構造体を用いてコマンドを定義していくのですが, この際に簡単にコマンドの設定 (help した際に出す文言など) を行えます。

4.2 Bubble Tea

例えば宅配物を登録する際, 宅配事業者の名前と追跡番号のどちらかの入力でも欠けている場合, 宅配物を追跡することができません。このため, これらの指定が欠けている際は, ユーザーに入力を促したいです。このとき, TUI によるリッチな入力フォームを作成することを考えました。

Bubble Tea^{*4} というフレームワークは, TUI を簡単に Go で作成することができます。Elm アーキテクチャに沿って実装するのが基本ですが, リポジトリに実装例が豊富なので目的に即したものもいい感じに変えるだけでリッチな CUI アプリケーションを作成することができます。

5 おわりに

今回はコマンドライン上で宅配物の追跡を一元管理するアプリケーションの概要と大まかな使用技術について紹介しました。皆さんもいい感じに生活を自動化していきましょう。

^{*4}<https://github.com/charmbracelet/bubbletea>

情報科学類誌

WORD

From College of Information Science

WORD編集部は除草剤を
撒いていません号

発行者	情報科学類長
編集長	伊藤祐聖
	筑波大学情報学群
	情報科学類 WORD 編集部
制作・編集	(第三エリアC棟212号室)

2023年12月25日 初版第1刷発行

(128部)