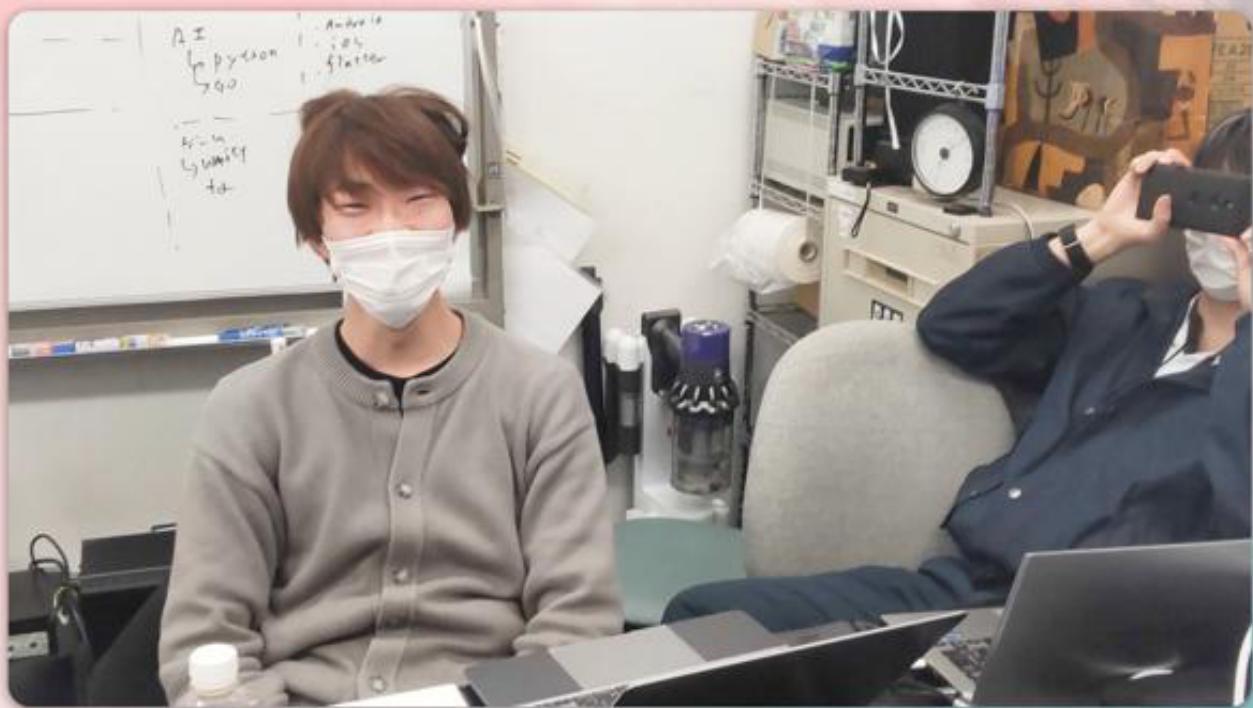


# シス研の技術本 テスト作成 表紙



# 目次

第 1 章	シス研というサークルについて	2
1.1	はじめに	2
1.2	シス研とは	2
1.3	この本について	5
1.4	まとめ	5
第 2 章	技術記事のまとめ	6
2.1	DiscordBot を作ってみよう	7
2.2	リポジトリ作成後に設定しておきたいこと	11
2.3	Gin × Neo4j × Docker で最短経路を返す API サーバを建てる	16
2.4	Next.js 13 触ってみた	29
第 3 章	シス研の日常	46
3.1	戦闘機に乗ろう	47
第 4 章	あとがき枠	50
4.1	本誌創刊に寄せて	50

# 1

## シス研というサークルについて

### 1.1 はじめに

初めまして！シス研会長の林です！はい、ここでみなさんシス研とはなんぞ？となっていると思うのでまずは自分が水先案内人となりましてシス研とこの本について解説していこうと思います。

#### 1.1.1 どんな話をするのか

シス研ってどんなサークル？どんな活動をしているの？この本はどういったもの？といったものを紹介していきます。それでは、さっそく行ってみましょう！！！

### 1.2 シス研とは

シス研は正式名称を「システム工学研究会」と言い、愛知工業大学公認の情報系サークルです。歴史は長く、2023年で創立47周年を迎え、かのAppleと同じ年となります！！  
シス研ではハッカソン出場をはじめとしたチーム開発、ゲーム作成、インフラの構築、運用などを行っています。

### 1.2.1 どんな活動をしているの？

シス研の主な活動はチーム開発とインフラ整備です。サークル全体としての開発物などはなく、それぞれがチームを組んでハッカソンに出場したりしています。

インフラ面では、部室に物理サーバを持っており、そこでシス研のホームページや各種サービスを公開しています。<sup>\*1\*2</sup>23年4月現在、大幅な工事を行なっておりごく一部のサービスのみ稼働しています（すみません）そのほかにもシス研主催のLT会・ハッカソンの開催、stech様<sup>\*3</sup>と共同でQiitaアドベントカレンダーへの参加もしています。<sup>\*4</sup>



▲ 図 1.1 部室の様子

### 1.2.2 2021～2022 年度の活動実績

- 2021 愛工大大学祭 工科展 最優秀賞
- 2022 技育博 参加
- 2022 Geekcamp vol8 優秀賞
- 2022 愛工大大学祭 工科展 瑞若賞
- 2022 技育展 出展

\*1 シス研ホームページ <https://set1.ie.aitech.ac.jp>

\*2 シス研紹介ページ <https://welcome.sysken.net>

\*3 stech様 HP <https://stech.careerselect.jp>

\*4 Advent Calendar 2022 <https://qiita.com/advent-calendar/2022/stech-ait-advent>

- 2022 愛工大大学祭 模擬店 最優秀賞
- 2022 HackU 春・夏 参加
- 2022 Geekcamp アドバンス 登壇
- 長期休暇中の LT 会、ハッカソン主催
- 各種勉強会の開催

### シス研の設備

- ブレードサーバ、ネットワーク機器
- デスクトップ PC
- iMac, MacBook
- iPhone, iPad
- Android 端末各種
- Raspberry Pi
- はんだ等の電子工作セット
- その他多数...



▲ 図 1.2 ブレードサーバ



▲ 図 1.3 ネットワーク機器



▲ 図 1.4 タブレット端末



▲ 図 1.5 Raspberry Pi

## 1.3 この本について

---

この本はシス研のメンバーが経験したこと、取り組んだことのアウトプットを目的としたものです。この本を通じて皆さんにはシス研のメンバーは具体的にどのような活動をしているのか知ってもらいたいと思います。

また、シス研として本を出すのは今回が初めてなのでどうか温かい目で見ていただけると幸いです。

## 1.4 まとめ

---

ここまでお話を来て来ましたが簡単にでもシス研について知ってもらうことはできたでしょうか？うまく伝えることができていたらとても嬉しいです。

次のページからはメンバーの記事本編になります！シス研初めての本をよろしくお願ひします！！

# 2

技術記事のまとめ

## 2.1 DiscordBot を作ってみよう

### 2.1.1 はじめに

はじめまして、suda です。私は Discord で人とチャットをしている時に同じ会話が頻繁に続き、これ Bot で返事をするようにしたら返事をする手間が省けるし面白いのでは？と思い Bot を作ることにしました。発想がひどいって！？まあでも自分の発想したものを形にすることが面白いことだと思うので今回はそこには目を瞑りましょう…もちろん自分が送ったメッセージに対して Bot に返答させることもできるので、自分だけのオリジナル DiscordBot を作ってみましょう！

### 2.1.2 何を作るのか

Discord のサーバで特定のメッセージが来たら、特定のメッセージを返す Discord の Bot を作ります。サンプルプログラムを参照したい方は以下の URL からご覧下さい。<sup>\*1</sup> 例えば自分が「仕事終わった」と言うと Bot が「お疲れ様」と返してくれます。

### 2.1.3 実行環境・使用技術

- Python 3.10.8

### 2.1.4 ローカル環境で Bot が動作するようにする

まずはローカル環境で Bot が動作するようにしてみます。

#### Bot の作成・管理をする

初めに、機能などはまだついていない Bot を Discord のポータルサイトから作成します。Discord の Bot の作り方（メモ）という記事の「1.Discord 上の Bot の作成」を見ながら Bot を作成してみて下さい。<sup>\*2</sup>

#### ファイルの作成

Bot を実行する Python ファイルを作ります。

<sup>\*1</sup> 今回作る DiscordBot のサンプルプログラム [https://github.com/sudamichiyo/Discord\\_Bot\\_sampleprogram](https://github.com/sudamichiyo/Discord_Bot_sampleprogram)

<sup>\*2</sup> Discord の Bot の作り方（メモ）<https://note.com/exteoi/n/nf1c37cb26c41>（参照 2023.3.29）

```
$ mkdir message_discord_bot  
$ cd message_discord_bot  
$ touch main.py
```

### discord.py の準備

ここからはライブラリ (discord.py) のドキュメントを見ながら環境構築をしていきます。<sup>\*3</sup> Python で Discord の API を操作するために必要なライブラリをインストールします。先ほど作成したディレクトリにアクセスして、以下のコマンドで discord.py をインストールします。

```
$ python3 -m pip install -U discord.py
```

次に、先ほど作成した main.py を以下のソースコードに書き換えます。

```
1 import discord  
2  
3 class MyClient(discord.Client):  
4     async def on_ready(self):  
5         print(f'Logged on as {self.user}!')  
6  
7     async def on_message(self, message):  
8         print(f'Message from {message.author}  
9                 : {message.content}')  
10  
11 intents = discord.Intents.default()  
12 intents.message_content = True  
13  
14 client = MyClient(intents=intents)  
15 client.run('my token goes here')
```

---

<sup>\*3</sup> discord.py ドキュメント <https://discordpy.readthedocs.io/ja/latest/intro.html#basic-concepts> (参照 2023.3.29)

ここで、以下のボットに関する 2 つの設定を Discord のポータルサイトから設定してください。

- ポータルサイトの「Bot」からトークンを取得する
- ポータルサイトの「Bot」の「MESSAGE CONTENT INTENT」を有効にする

'my token goes here' は取得した Bot のアクセストークンを書きます。以上の設定が終わったら python3 main.py を実行すると、Bot のサーバが立ち上がります。Bot のいるサーバの任意のチャネルでメッセージを投稿すると、コマンドライン上に「書いた人」と「メッセージ」がそのまま出力されます。

### 環境変数の設定

ソースコードに直接トークンを書いてしまうと、Github でソースコードをホスティングするときにトークンキーが他の人にバレてしまいます。これを防ぐために.env ファイルを作成して、その中に Discord のアクセストークンを書きます（下記参照）。

```
1 DISCORD_TOKEN='My token goes here'
```

Python の中で.env ファイルに書かれている変数を取得するために dotenv というライブラリを使用します。以下のようにインストールします。

```
$ pip3 install python-dotenv
```

インストール後に main.py に下記のコードを付け加えて下さい。

main.py の import discord と class MyClient の間に以下のコードを追加します。

```
1 import os
2 from dotenv import load_dotenv
3 load_dotenv()
```

そして、最後の行を以下のように書き換えて下さい。

```
1 client.run(os.environ['DISCORD_TOKEN'])
```

書き換えたあとに python3 main.py を実行すると、先ほどと同じようにメッセージの受け取りをしてくれるサーバーサイドアプリケーションが立ち上がります。

Bot が特定のワードに反応して、特定のメッセージを返答する機能をつける

プログラムを起動して正常にサーバーサイドアプリケーションがメッセージを受け取れるようになったら、Bot が特定のワードに反応して、特定のメッセージを返答する機能をつけていきます。Bot に機能をつけるには上記のソースコードの 8 行目と 10 行目の間に以下のコードを付け足していきます。

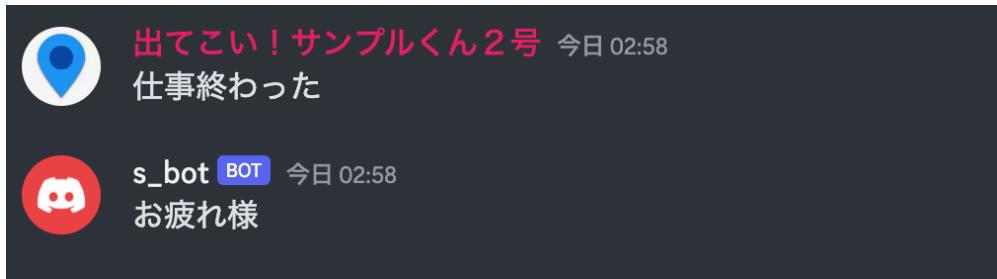
```
1 # メッセージを書いた人が Bot なら処理終了
2 if message.author.bot:
3     return
4 channel = message.channel
5 if message.content == '仕事終わった':
6     await channel.send('お疲れ様')
```

付け足したコードの解説をしていきます。

2,3 行目でメッセージを書いた人が Bot なら処理を終了させています。

4 行目でメッセージが投稿されたチャンネル取得しています。

5 行目の message.content はメッセージの内容で、今回の場合「仕事終わった」というメッセージをチャンネルに投稿すると、メッセージが投稿されたチャンネルに Bot が「お疲れ様」と返答します。



▲ 図 2.1 動作例

### 2.1.5 まとめ

今回は Discord の Bot の作り方を説明しました。上記の「仕事終わった」や「お疲れ様」に当たる部分を変えたりして自分好みに改良してみて下さい。

## 2.2 リポジトリ作成後に設定しておきたいこと

### 2.2.1 はじめに

こんにちは。hihumikan です。

本チャプターでは「リポジトリ作成後に設定しておきたいこと」をご紹介します。私自身が次回プロジェクト開始する際に、こういった設定や技術を使うだろうというものをまとめました。

ただし、これら全ての内容をプロジェクトに適用出来るものではないため、ご利用の環境や用途に合わせて利用していただければと思います。

#### 対象読者

対象読者は、Git/GitHub を利用した開発を行う初学者の方を想定しています。

### 2.2.2 ファイルの設定

リポジトリ作成後に設定しておきたい「ファイルの設定」についてご紹介します。

#### .gitignore

.gitignore は、Git による管理から除外したいファイルやディレクトリを指定するための設定ファイルです。

例えば、MacOS の場合、ディレクトリ毎に.DS\_Store というファイルが自動的に生成されます。このファイルは、ディレクトリの meta 情報を記録しますが、通常の開発において共有する必要がないため、Git の管理対象外としておくのが望ましいです。

また、.env などの環境変数を利用してプログラムを動かす場合、.env ファイルには、パスワードや秘密鍵などの情報が含まれているのが多いため、これも Git の管理対象外としておくのが望ましいです。

リスト 1.1 のようなテキストファイルを Git の管理下に置くだけで、Git から管理対象外として扱われます。

#### リスト 1.1 .gitignore

```
1 .DS_Store  
2 .env
```

リポジトリを共有する場合、.gitignore ファイルを置いておくだけで、開発メンバー全員が同じ設定を利用できるため、必要なファイルだけが Git の管理下に置かれるようになります。

## .gitattributes

.gitattributes は、特定のファイルに対して Git の挙動を変更するための設定ファイルです。主に、改行コードやファイルの文字コードを指定するために利用されます。

改行コードを指定する理由としては、Windows と MacOS では改行コードが異なる点が挙げられます。Git で管理されているファイルの改行コードが一致しないと、差分が発生してしまい、想定した挙動と異なる動作をする可能性があります。それを防ぐために、.gitattributes に改行コードを明示しておくことで、安全に開発を進めれます。

リスト 1.2 のように、ファイルの拡張子に対して、改行コードを指定が出来ます。

### リスト 1.2 .gitattributes

```
1 * text=auto  
2 *.sh text eol=lf
```

これも.gitignore と同様に.gitattributes 共有するだけで、開発メンバー全員が同じ設定を利用できます。

## Makefile

Makefile は、コマンドをまとめて実行するための設定ファイルです。

利点として、開発メンバー全員が同じコマンドを実行出来る所にあります。環境構築やテストの実行など、手順が複雑な作業を一人一人が実行した場合、手順の違いによるエラーが発生する可能性があります。それらを防ぐために、Makefile にまとめておくことで、開発メンバー全員が同じコマンドを実行でき、人的ミスを防げます。

Makefile の例としては、リスト 1.3 のようなものです。

### リスト 1.3 Makefile

```
1 up: ## API とデータベースを起動
2   docker compose -f docker-compose-db.yml -p db up -d
3   docker compose -f docker-compose-api.yml -p api up -d
4
5 build: ## サービスの構築
6   docker compose -f docker-compose-db.yml -p db build
7   docker compose -f docker-compose-api.yml -p api build
8
9 stop: ## サービスを停止
10  docker compose -f docker-compose-db.yml -p db stop
11  docker compose -f docker-compose-api.yml -p api stop
12
13 kill: ## サービスを強制停止
14  docker compose -f docker-compose-db.yml -p db kill
15  docker compose -f docker-compose-api.yml -p api kill
16
17 down: ## サービスの停止とコンテナの削除
18  docker compose -f docker-compose-db.yml -p db down
19  docker compose -f docker-compose-api.yml -p api down
20
21 restart: ## サービスの再起動
22  docker compose -f docker-compose-db.yml -p db restart
23  docker compose -f docker-compose-api.yml -p api restart
```

これらを実行する場合、シェルに

```
$ make up
```

と入力するだけで、長いコマンドであった 2,3 行目のコマンドが簡単に実行されます。リスト 1.3 の例は簡単なものですぐ、他にも Makefile 内に変数を定義できるため、変更が必要な箇所を変数に置き換えることで、コマンドの変更を容易に行えます。

### 2.2.3 インフラ回り

#### GitHub Actions

Github Actions は、GitHub 上で動作する CI/CD ツールです。簡単に言えば、GitHub リポジトリに関連するイベントに応じて、あらかじめ定義しておいたワークフローを仮想マシン上で実行するものです

用途としては、コードの静的解析やテストの実行、デプロイなどが挙げられます。その他にも、Pull Request に対して、テスト内容をコメントしてくれるなどの GitHub 上での機能を利用できます。

利用方法としては、リスト 1.4 のように、.github/workflows ディレクトリを作成し、その中に設定ファイルを作成します。

リスト 1.4 .github/workflows/pr.yml

```
1 on:
2   pull_request:
3     types: [opened]
4   name: Pull Request
5   jobs:
6     assignAuthor:
7       name: Assign author to PR
8       runs-on: ubuntu-latest
9     steps:
10       - name: Assign author to PR
11         uses: technote-space/assign-author@v1
```

上記の例では、Pull Request が作成された際に、Pull Request の作成者を Assignee に設定する設定ファイルの例です。

その他にも、リスト 1.5 のように、ssh 鍵を設定することで、リモートサーバーにアクセスができます。

### リスト 1.5 .github/workflows/deploy.yml

```
1 name:CI
2 on:
3   push:
4     branches:
5       - main
6 jobs:
7   deploy:
8     runs-on: ubuntu-latest
9     steps:
10    - uses: actions/checkout@v2
11    - name: Install SSH Key for Deploy
12      uses: appleboy/ssh-action@master
13      with:
14        key: ${{ secrets.SK }}
15        host: ${{secrets.SSH_HOST}}
16        username: ${{secrets.SSH_USERNAME}}
17        port: ${{secrets.SSH_PORT}}
18        script: |
19          git pull
```

この他にも、コードを自動で整形して commit してくれるツールなどの様々なツールが存在します。調べてみると面白いかもしれません。

#### 2.2.4 おわりに

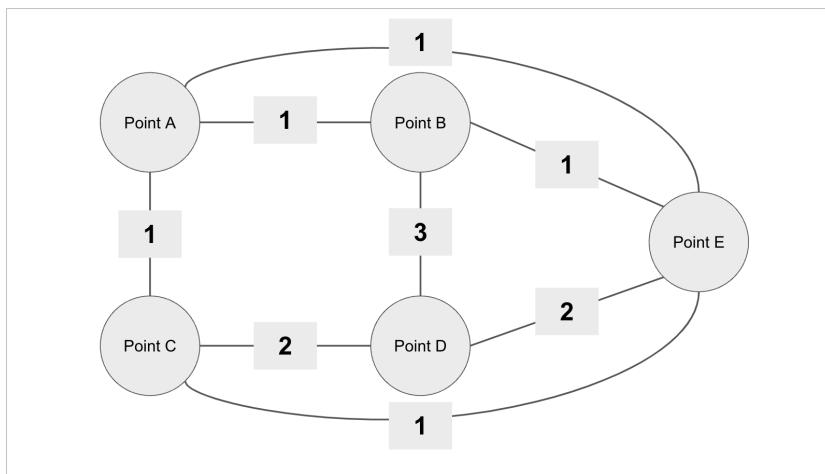
本チャプターでは「リポジトリ作成後に設定しておきたいこと」をご紹介しました。ご紹介したのは一部分ですが、これらを設定しておくことで、開発の効率化や、開発メンバーのミスを防げます。ぜひ、設定してみてください。

## 2.3 Gin × Neo4j × Docker で最短経路を返す API サーバを建てる

### 2.3.1 はじめに

このセクションでは、現在最もメジャーなグラフ DB である Neo4j と Github でも多くのスターを獲得している Go 言語の Web フレームワーク、Gin を用いて API サーバーを構築していきます。また、実行環境統一のため Docker を用います。

### 2.3.2 今回扱うデータの図



### 2.3.3 ディレクトリ構造

#### ディレクトリ構造

```
├── build
│   ├── Docker
│   │   ├── go
│   │   │   └── Dockerfile
│   │   └── neo4j
│   │       ├── Dockerfile
│   │   └── volumes
│   │       ├── import
│   │       │   ├── done
│   │       │   ├── points.csv
│   │       │   └── route.csv
│   │       └── script
│   │           └── import_data.sh
│   └── docker-compose.yml
└── server
    ├── config
    │   ├── config.go
    │   └── environments
    │       └── neo4j.yml
    ├── controllers
    │   └── coordinate_controller.go
    ├── db
    │   └── neo4j.go
    ├── go.mod
    ├── go.sum
    ├── main.go
    ├── models
    │   └── coordinate.go
    ├── router
    │   └── router.go
    └── sample.http
```

### 2.3.4 Neo4j に最初にインポートするファイル

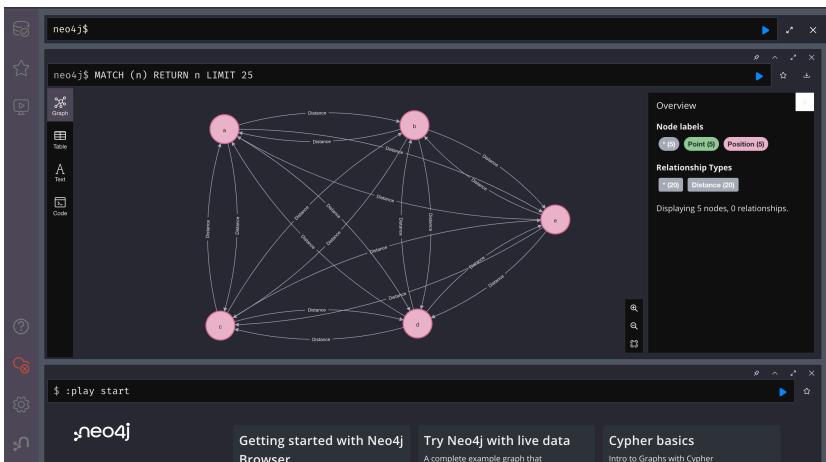
point.csv

```
1 point_id:ID,point_name,:LABEL
2 a,PointA,Point;Position
3 b,PointB,Point;Position
4 c,PointC,Point;Position
5 d,PointD,Point;Position
6 e,PointE,Point;Position
```

route.csv

```
:START_ID,:END_ID,:TYPE,cost:int
b,a,Distance,1
c,b,Distance,3
d,c,Distance,2
e,d,Distance,2
c,a,Distance,1
d,a,Distance,2
e,b,Distance,1
e,a,Distance,1
c,e,Distance,1
d,b,Distance,3
a,b,Distance,1
b,c,Distance,3
c,d,Distance,2
d,e,Distance,2
a,c,Distance,1
a,d,Distance,2
b,e,Distance,1
a,e,Distance,1
e,c,Distance,1
b,d,Distance,3
```

これらを Neo4j にインポートすることで、先ほどの図を表現することができます。



無事できていますね。

neo4j.go

```

1 version: "3"
2 services:
3   go:
4     container_name: NEO4JAPI_G0
5     build:
6       context: ./docker/go
7       dockerfile: Dockerfile
8     stdin_open: true
9     tty: true
10    volumes:
11      - ../server:/server
12    ports:
13      - 8080:8080
14    networks:
15      app_net:
16        ipv4_address: 192.168.0.1
17    depends_on:
18      - "neo4j"
19
20 neo4j:
```

```

21   container_name: NE04JAPI_NE04J
22   build:
23     context: ./Docker/neo4j
24     dockerfile: Dockerfile
25   restart: always
26   ports:
27     - 57474:7474
28     - 57687:7687
29   volumes:
30     - ./Docker/neo4j/volumes/data:/data
31     - ./Docker/neo4j/volumes/logs:/logs
32     - ./Docker/neo4j/volumes/conf:/conf
33     - ./Docker/neo4j/volumes/import:/import
34     - ./Docker/neo4j/volumes/script:/script
35   environment:
36     - NEO4J_AUTH=neo4j/admin
37     - EXTENSION_SCRIPT=/script/import_data.sh
38   networks:
39     app_net:
40       ipv4_address: 192.168.0.2
41
42 networks:
43   app_net:
44     driver: bridge
45     ipam:
46       driver: default
47       config:
48         - subnet: 192.168.0.0/24

```

volumes はデータベースに入れるデータ、ログをバインドするための場所を指定しています。environment ではユーザーとパスワード、最初に読み込んでもらうシェルスクリプトの場所を指定します。

go 関連の Dockerfile。

## Dockerfile

```
1 # go バージョン
2 FROM golang:1.19.3-alpine
3 # アップデートと git のインストール
4 RUN apk add --update && apk add git
5 # app ディレクトリの作成
6 RUN mkdir /server
7 # ワーキングディレクトリの設定
8 WORKDIR /server
9 # ホストのファイルをコンテナの作業ディレクトリに移行
10 ADD . /server
11 # main.go を実行
12 CMD ["go", "run", "main.go"]
```

Neo4j 関連の Dockerfile。

## Dockerfile

```
1 # go バージョン
2 FROM golang:1.19.3-alpine
3 # アップデートと git のインストール
4 RUN apk add --update && apk add git
5 # app ディレクトリの作成
6 RUN mkdir /server
7 # ワーキングディレクトリの設定
8 WORKDIR /server
9 # ホストのファイルをコンテナの作業ディレクトリに移行
10 ADD . /server
11 # main.go を実行
12 CMD ["go", "run", "main.go"]
```

csv ファイルを読ませるためのシェルスクリプト。

```
import_data.sh
```

```
1 #!/bin/bash
2 set -euC
3
4 # EXTENSION_SCRIPT はコンテナが起動するたびにコールされるため、
5 # import 処理が実施済かフラグファイルの有無をチェック
6 if [ -f /import/done ]; then
7     echo "Skip import process"
8     return
9 fi
10
11 # データを全削除
12 echo "delete database started."
13 rm -rf /data/databases
14 rm -rf /data/transactions
15 echo "delete database finished."
16
17 # CSV データのインポート
18 echo "Start the data import process"
19 neo4j-admin import \
20   --nodes=/import/points.csv \
21   --relationships=/import/route.csv
22 echo "Complete the data import process"
23
24 # import 処理の完了フラグファイルの作成
25 echo "Start creating flag file"
26 touch /import/done
27 echo "Complete creating flag file"
```

### 2.3.5 Go のソースコード

config ディレクトリ

```
config.go
```

```
1 package config
2
3 import (
4     "github.com/spf13/viper"
5 )
6
7 var n *viper.Viper
8
9 func init() {
10    n = viper.New()
11    n.SetConfigType("yaml")
12    n.SetConfigName("neo4j")
13    n.AddConfigPath("config/environments/")
14 }
15
16 func GetNeo4jConfig() *viper.Viper {
17    if err := n.ReadInConfig(); err != nil {
18        return nil
19    }
20    return n
21 }
```

このファイルで neo4j.yaml の環境変数を読み込みます。

```
neo4j.yml
```

```
1 neo4j:
2   user: neo4j
3   password: admin
4   uri: neo4j://192.168.176.1:57687
```

Neo4j と接続するための環境変数です。

## db ディレクトリ

### neo4j.go

```
1 package db
2
3 import (
4     "log"
5
6     "neo4japi/server/config"
7
8     "github.com/neo4j/neo4j-go-driver/v4/neo4j"
9 )
10
11 func GetDriverAndSession() neo4j.Session {
12     n := config.GetNeo4jConfig()
13     dr, err := neo4j.NewDriver(n.GetString("neo4j.uri"),
14         neo4j.BasicAuth(n.GetString("neo4j.user"), n.
15             GetString("neo4j.password"), ""))
16     if err != nil {
17         log.Fatal(err)
18     }
19     ses := dr.NewSession(neo4j.SessionConfig{AccessMode
20         : neo4j.AccessModeRead})
21     return ses
22 }
```

このファイルで Neo4j と接続し、セッションを返すようにします。

## models ディレクトリ

### coordinate.go

```
1 package models
2
3 import (
4     "fmt"
5     "log"
6
7     "neo4japi/server/db"
8
9     "github.com/neo4j/neo4j-go-driver/v4/neo4j"
10 )
11
12 type Route struct {
13     Position string `json:"point"`
14 }
15
16 func FindRoute(fr, to string) []*Route {
17     var r []*Route
18     ses := db.GetDriverAndSession()
19     defer ses.Close()
20     cyp := fmt.Sprintf(`
21         MATCH (from:Position {point_name: "%s"}) , (to:
22             Position {point_name: "%s"})
23             path=allShortestPaths ((from)-[distance:
24             Distance*]->(to))
25             WITH
26                 [position in nodes(path) | position.
27                  point_name] as name,
28             REDUCE(totalMinutes = 0, d in distance |
29                 totalMinutes + d.cost) as 所要時
30             間
```

```

26      RETURN name
27      ORDER BY 所要時間
28      LIMIT 10;
29      ', fr, to)
30
31  _, err := ses.ReadTransaction(func(transaction
neo4j.Transaction) (interface{}, error) {
32      result, err := transaction.Run(cyp, nil)
33      if err != nil {
34          return nil, err
35      }
36      if result.Next() {
37          name, _ := result.Record().Get("name")
38          nameAr := name.([]interface{})
39
40          for i := 0; i < len(nameAr); i++ {
41              r = append(r, &Route{nameAr[i].(string)})
42          }
43      }
44      return nil, result.Err()
45  })
46  if err != nil {
47      log.Fatal(err)
48  }
49  return r
50 }

```

出発地点と目的地を受け取ることで Neo4j に Cypher というクエリ言語を用いて最短経路を導出してもらいます。

## controllers ディレクトリ

### coordinate.controller.go

```
1 package controllers
2
3 import (
4     "net/http"
5
6     "github.com/gin-gonic/gin"
7     "neo4japi/server/models"
8 )
9
10 func RouteSearch(c *gin.Context) {
11     fr := c.Query("fr")
12     to := c.Query("to")
13     res := models.FindRoute(fr, to)
14     c.JSON(http.StatusOK, res)
15 }
```

GET リクエストで受け取ったパラメータを models で作成した関数に渡します。

## router ディレクトリ

### router.go

```
1 package router
2
3 import (
4     "github.com/gin-gonic/gin"
5     "neo4japi/server/controllers"
6 )
7
8 func Init() {
9     r := gin.Default()
10    r.GET("/coordinate", controllers.RouteSearch)
11    r.Run()
12 }
```

ルーティング先を定義します。

#### 実行ファイル

coordinate\_controller.go

```
1 package main
2
3 import (
4     "neo4japi/server/router"
5 )
6
7 func main() {
8     router.Init()
9 }
```

#### 2.3.6 実行方法

docker compose up

このコマンドを実行することで Neo4j サーバーと Gin サーバーが立ち上がります。

#### 実行確認

sample.http

```
1 GET http://localhost:8080/coordinate?fr=PointB&to=PointE
```

今回は Vscode の拡張機能である REST Client を用いて実行確認を行います。

ここで、 coordinate?fr=PointB&to=PointE の部分を自分の好きな地点にしてリクエストを送ると、最短経路が返されます。

#### 2.3.7 おわりに

このチャプターではグラフ DB と Go 言語を用いた API サーバーの建て方を説明しました。Gin、Neo4j は奥が深いので、もし気になった方はぜひ自分で調べて触ってみてください。

## 2.4 Next.js 13 触ってみた

### 2.4.1 はじめに

10月後半に行われた Next.js Conf 2022 で発表された Next.js 13 を実際に触ってみたのでその内容を書きます。当初は全体を網羅して書く予定だったのですが量が多くかったの少し絞ってます。対象読者としてある程度 React や Next.js を触っている人を対象としています。この記事は半年程前に書いたため現在と異なっている場合があります。執筆時の Next のバージョンは 13.0.5 です。

### 2.4.2 プロジェクト作成からサーバ起動

TypeScript と ESLint を入れるか聞かれるので入れる。

```
1 $ npx create-next-app@latest --ts
```

pages ディレクトリを削除。

```
1 $ rm -rf pages
```

app ディレクトリを作る。

```
1 $ mkdir app
```

app ディレクトリは実験段階の機能なので, next.config.js を変更する。

next.config.js

```
1 const nextConfig = {
2   reactStrictMode: true,
3   swcMinify: true,
4   experimental: {
5     appDir: true,
6   },
7 };
```

app/page.tsx を作り、以下のようにする。

### app/page.tsx

```
1  export default function Page() {  
2      return <h1>Hello, Next.js!</h1>;  
3  }
```

ローカルサーバ起動。

```
1  $ npm run dev
```

ブラウザで `http://localhost:3000/` にアクセスすると、Hello, Next.js! が表示される。



**Hello, Next.js!**

▲ 図 2.2 Hello,Next.js

### 2.4.3 Layout と Head

サーバを起動すると自動的に app ディレクトリに layout.tsx、head.tsx というファイルが作られていきました。

next/head を使って各ページファイルに head を定義していたのが head.tsx に書けるようになったみたいですね。

### head.tsx

```
1  export default function Head() {  
2      return (  
3          <>
```

```
4      <title></title>
5      <meta content="width=device-width,initial-scale=1" name="viewport" />
6      <link rel="icon" href="/favicon.ico" />
7      </>
8    )
9 }
```

ページの共通レイアウトを定義するファイル。

```
1 export default function RootLayout({
2   children,
3 }: {
4   children: React.ReactNode
5 }) {
6   return (
7     <html>
8       <head />
9       <body>{children}</body>
10      </html>
11    )
12 }
```

今まででは`_app.tsx`などに下記のようにレイアウトを定義していました。

`_app.tsx`

```
1 <Layout>
2   <Component {...pageProps} />
3 </Layout>
```

この方法だとページごとにレイアウトを変えられないです。なのでページごとにレイアウトを変えたい場合は`getLayout`を用いる必要がありました。Next.js 13 ではレイアウトを変

えたいページがあるディレクトリに layout.tsx を置けばページごとにレイアウトを変えられるようになりました。

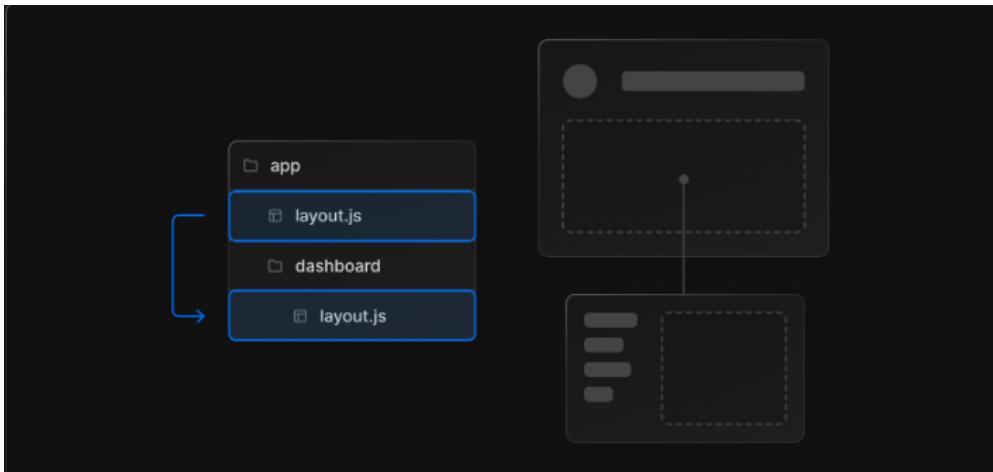
ここではダッシュボードページのレイアウトの場合を考えます。app ディレクトリの中で dashboard ディレクトリを作成して以下のように layout.tsx を配置するだけです。

#### app/dashboard/layout.tsx

```
1  export default function DashboardLayout({  
2      children,  
3  }: {  
4      children: React.ReactNode;  
5  }) {  
6      return <subsection>{children}</subsection>;  
7  }
```

少し疑問に思ったのが dashboard ディレクトリの page.tsx では RootLayout は呼ばれず DashboardLayout のみが呼ばれるのかと思っていました。しかし、実際には入れ子構造で呼び出されるようです。

公式の画像がわかりやすいので貼っておきます。



▲ 図 2.3 layout.js の適用範囲

次の機能に行く前に dashboard ディレクトリに page.tsx も作っておきます。

app/dashboard/layout.tsx

```
1  export default function DashBoardPage() {  
2      return <h1>DashBoard Page</h1>;  
3  }
```

違いがわかりやすいように他のファイルも変更します。

app/layout.tsx

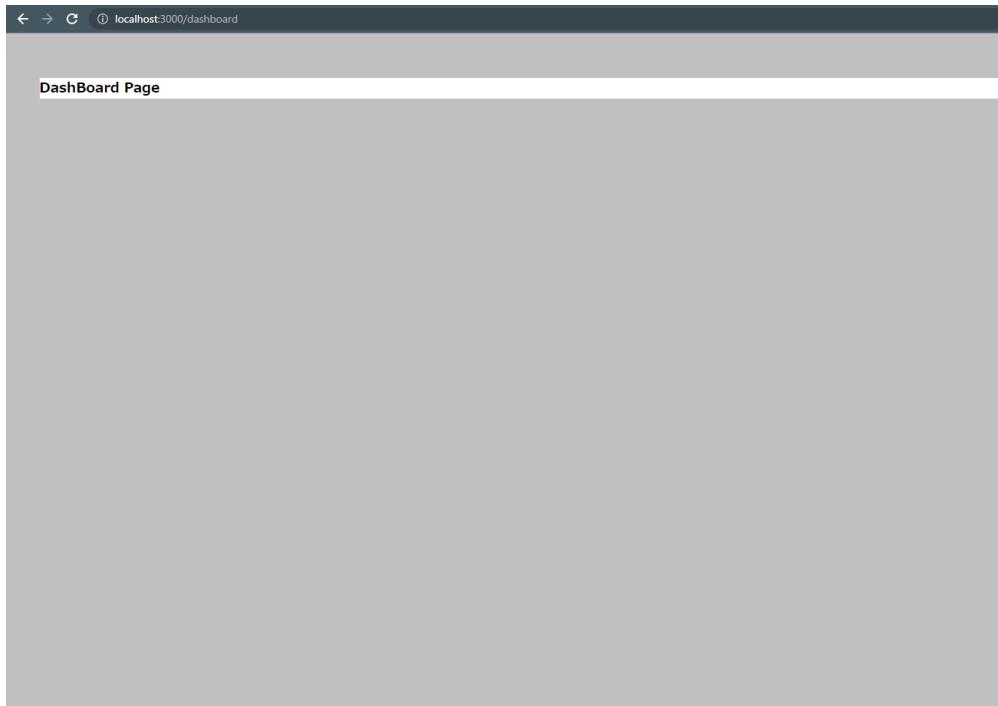
```
1  export default function RootLayout({  
2      children,  
3  }: {  
4      children: React.ReactNode;  
5  }) {  
6      return (  
7          <html>  
8              <head />  
9              <body  
10                 style={{  
11                     backgroundColor: '#COCOCO',  
12                     padding: '50px',  
13                 }}  
14             >  
15             {children}  
16         </body>  
17     </html>  
18 );  
19 }
```

app/dashboard/layout.tsx

```
1  export default function DashboardLayout({
```

```
2         children,
3     }: {
4         children: React.ReactNode;
5     }) {
6         return (
7             <subsection
8                 style={{
9                     backgroundColor: 'white',
10                }}
11            >
12            {children}
13        </subsection>
14    );
15 }
```

ページの見た目が画像のようになってればOK。



▲ 図 2.4 DashBoard ページ

#### 2.4.4 React Server Components

React Server Components(RSC) は React18 で追加された機能でクライアントとサーバ側が協調してアプリケーションをレンダリングできる機能です。これによりコンポーネントごとに最適なレンダリング方法を選択できるようになります。例えばデータの取得はサーバ側でを行い、ユーザの操作によって変わる部分はクライアント側でレンダリングできます。これも公式ドキュメントの図がわかりやすいので貼っておきます。

What do you need to do?	Server Component	Client Component
Fetch data. Learn more.	✓	⚠
Access backend resources (directly)	✓	✗
Keep sensitive information on the server (access tokens, API keys, etc)	✓	✗
Keep large dependencies on the server / Reduce client-side JavaScript	✓	✗
Add interactivity and event listeners ( <code>onClick()</code> , <code>onChange()</code> , etc)	✗	✓
Use State and Lifecycle Effects ( <code>useState()</code> , <code>useReducer()</code> , <code>useEffect()</code> , etc)	✗	✓
Use browser-only APIs	✗	✓
Use custom hooks that depend on state, effects, or browser-only APIs	✗	✓
Use <a href="#">React Class components</a>	✗	✓

▲ 図 2.5 Sever Component と Client Components の違い

また SSRとの違いとしてクライアント側の JavaScript の量を減らせる点です。SSR の場合ハイドレーション（サーバ側で生成した DOM とクライアントで生成した DOM を合成する）というステップがありページを早く表示できてもクライアント側でも同じ処理が走るため JavaScript の量は同じでした。RSC はサーバ側でレンダリングした後残りをクライアント側でレンダリングします。これによってクライアント側に送信される JavaScript の量を減らせます。

## 2.4.5 サーバーコンポーネントでデータを取得

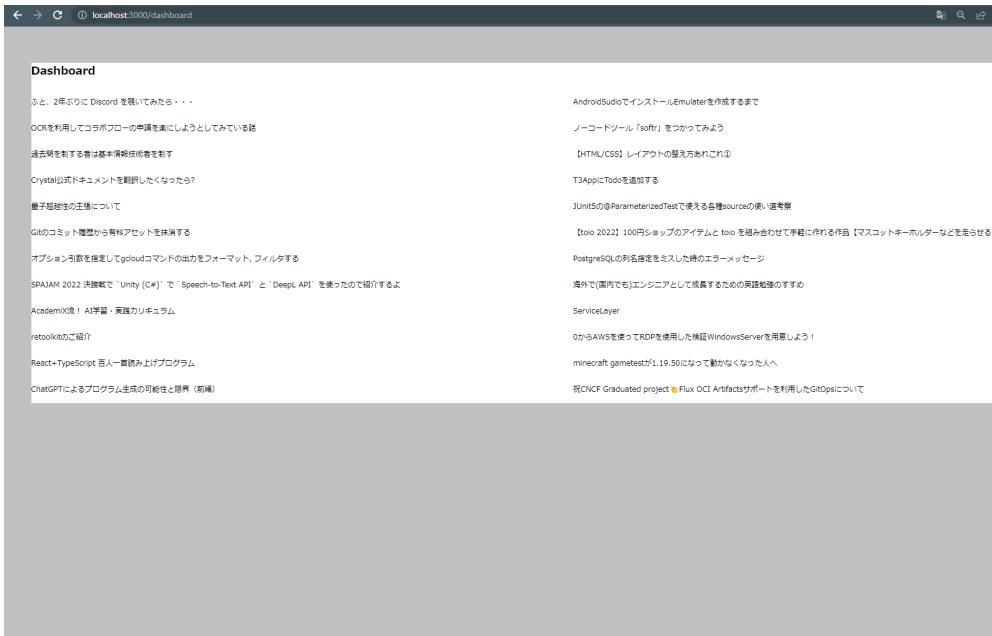
app ディレクトリ内のコンポーネントはデフォルトだとサーバコンポーネントになっています。以下のコードはサーバサイドで qiita の記事リストを取得して表示するものです。dashboard/page.tsx を書き換えます。

```
dashboard/page.tsx

1  type Article = {
2      id: number;
3      title: string;
4  };
5
6  async function getArticle(): Promise<Article[]> {
7      const res = await fetch('https://qiita.com/api/v2/items?page=1&
8          per_page=24');
9
10     if (!res.ok) {
11         throw new Error('Failed to fetch data');
12     }
13
14     return res.json();
15
16 }
17
18
19 export default async function DashBoardPage() {
20     const articles = await getArticle();
21
22     return (
23         <div>
24             <h1>Dashboard</h1>
25             <div
26                 style={{
27                     display: 'flex',
28                     flexDirection: 'column',
29                     ...
30                 }
31             <ul>
32                 {articles.map(article => (
33                     <li>{article.title}</li>
34                 ))}
35             </ul>
36         </div>
37     )
38 }
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
311
312
313
313
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
```

```
26         flexWrap: 'wrap',
27         height: '50vh',
28     )}
29     >
30     {articles?.map((article) => (
31         <div
32             key={article.id}
33             style={{{
34                 display: 'flex',
35                 gap: '10px',
36             }}}
37         >
38             <p>{article.title}</p>
39             </div>
40         )));
41     </div>
42     </div>
43 );
44 }
```

画像のように表示される。



▲ 図 2.6 API からデータ取得後のページ

サイトにカーソルを合わせて右クリックしてページのソースを表示をクリックしてみましょう。あらかじめデータが入った状態でサーバから送られてくるのでページソースに記事データが表示されています。



▲ 図 2.7 ページのソース表示

#### 2.4.6 ローディング UI 表示

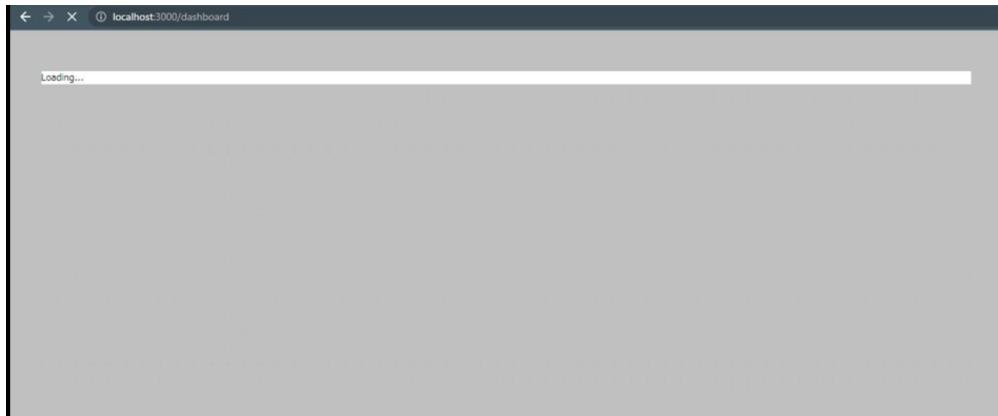
次はローディング UI を表示する機能です。dashborad ディレクトリに loading.tsx を作成します。

loading.tsx

```
1  export default function Loading() {  
2      return <p>Loading...</p>;
```

```
3 }
```

データの取得が終わり、page コンポーネントがレンダリングされるまでの間は Loading コンポーネントが表示されます。



▲ 図 2.8 ローディング UI

これは React18 で追加された Suspense という機能が使われていてます。Suspense について説明するとコンポーネントが表示されるまでの状態を指定できるコンポーネントです。非同期的なコンポーネントの場合レンダリングに時間がかかるためその間に何を表示させるかを Suspense を使うと指定できるようになります。

具体的な使用例を上げます。下のコードはデータフェッチャライブラリ React Query を使ったデータ取得と表示のサンプルです。

```
1 import { QueryClient, QueryClientProvider, useQuery } from 'react
  -query';
2
3 const queryClient = new QueryClient();
4
5 export default function App() {
6   return (
7     <QueryClientProvider client={queryClient}>
8       <Example />
9     </QueryClientProvider>

```

```
10      );
11  }
12  export function Loading() {
13      return <p>Loading...</p>;
14  }
15
16  function Example() {
17      const { isLoading, data } = useQuery('repoData', () =>
18          fetch('https://api.github.com/repos/tannerlinsley/react-query')
19              .then(
20                  (res) => res.json()
21              )
22      );
23
24      if (isLoading) return <Loading />;
25
26      return (
27          <div>
28              <h1>{data.name}</h1>
29              <p>{data.description}</p>
30          </div>
31      );
32  }
```

Suspense を使えば下のコードに置き換えられます。

```
1 import { Suspense } from 'react';
2 import { QueryClient, QueryClientProvider, useQuery } from 'react-
query';
3
4 const queryClient = new QueryClient();
5
```

```

6 export default function App() {
7   return (
8     <QueryClientProvider client={queryClient}>
9       {/* コンポーネントででラップ suspense */}
10      <Suspense fallback={<Loading />}>
11        <Example />
12      </Suspense>
13    </QueryClientProvider>
14  );
15 }
16 export function Loading() {
17   return <p>Loading...</p>;
18 }
19
20 function Example() {
21   const { data } = useQuery('repoData', () =>
22     fetch('https://api.github.com/repos/tannerlinsley/react-query')
23       .then(
24         (res) => res.json()
25       )
26   );
27   //ローディングプロパティによる表示分岐の削除
28   return (
29     <div>
30       <h1>{data.name}</h1>
31       <p>{data.description}</p>
32     </div>
33   );
34 }

```

変更点は Example コンポーネントを Suspense コンポーネントでラップしているのと、Example コンポーネントの中の isLoading プロパティを使った表示切替の部分が消えている

ところです。Suspense のいいところはデータ取得をするコンポーネントの中で表示を切り替える処理を書く必要がない所です。これによってより宣言的なコードになりました。またコンポーネントの責務の観点から見てもローディング完了時の表示だけでよくシンプルになっています。

Next.js 13 では loading.tsx を置いてあげると Next.js 側がそれを読み取り Page コンポーネントを Suspense コンポーネントでラップしてくれるようです。普通に書くと以下のようになります。

```
1 <Layout>
2   <Headr/>
3   <SideNav/>
4   <Suspense fallback={<Loading />}>
5     <DashBoardPage />
6   </Suspense>
7 </Layout>
```

ディレクトリ内に loading.tsx を配置すると、自動的にこのコードを記述した動作が実現されます。

#### 2.4.7 エラーハンドリング

次にエラーハンドリングを見ていきます。dashboard ディレクトリに error.tsx を作成します。

dashboard/error.tsx

```
1 'useClient';
2
3 import { useEffect } from 'react';
4
5 export default function Error({
6   error,
7   reset,
8 }: {
9   error: Error;
10  reset: () => void;
11}) {
```

```

12  useEffect(() => {
13    // Log the error to an error reporting service
14    console.error(error);
15  }, [error]);
16
17  return (
18  <div>
19    <p>Something went wrong!</p>
20    <button onClick={() => reset()}>Reset error boundary</button>
21  </div>
22 );
23 }

```

公式ドキュメントによると error.tsx はクライアントコンポーネントである必要があり useEffect; と書けばクライアントコンポーネントして利用するできるようです。loading.tsx と同じようにファイルを置いておけば自動的に Page をネストしてエラーハンドリングをしているようです。

#### 普通に書いた場合

dashboard/error.tsx

```

1 <Layout>
2   <Header/>
3   <SideNav/>
4   <ErrorBoundary fallback={<Loading />}>
5     <DashBoardPage />
6   </ErrorBoundary>
7 </Layout>

```

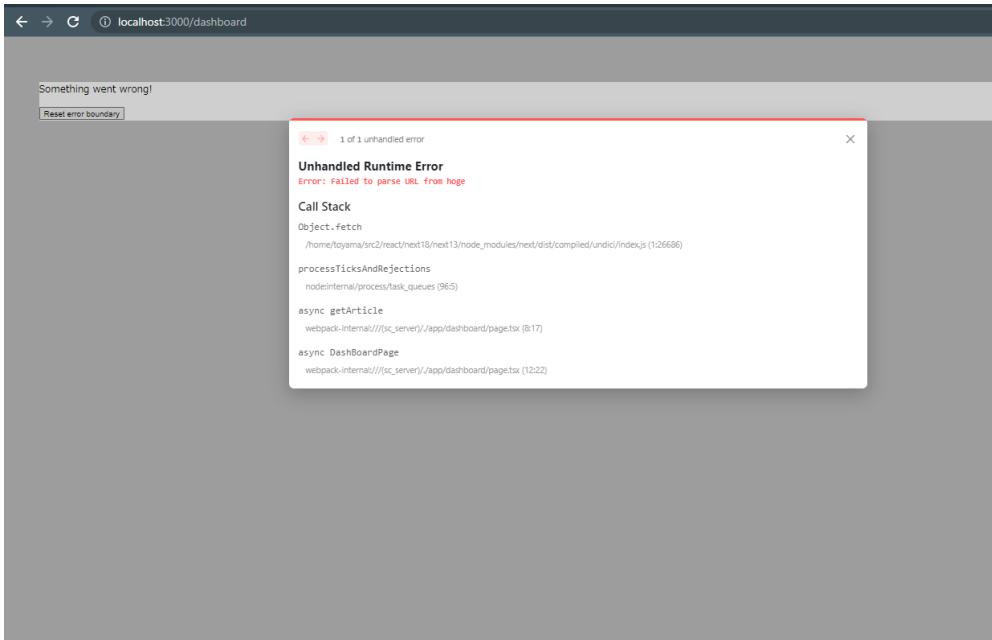
ただこれを見てわかる通り page をラップしてるので同階層のコンポーネントのエラーハンドリングはできない感じですね。したいならより上の階層で ErrorBoundary を使う必要があります。

実際にコードを書き換えてエラーを出してみます。存在しない URL 'hoge' を指定し getArticle でしていたエラーハンドリングを削除しています。

## dashboard/page.tsx

```
1
2 'useClient';
3
4 import { useEffect } from 'react';
5
6 export default function Error({
7   error,
8   reset,
9 }: {
10   error: Error;
11   reset: () => void;
12 }) {
13   useEffect(() => {
14     // Log the error to an error reporting service
15     console.error(error);
16   }, [error]);
17
18   return (
19     <div>
20       <p>Something went wrong!</p>
21       <button onClick={() => reset()}>Reset error boundary</button>
22     </div>
23   );
24 }
```

エラーの内容とエラーページが表示されていますね。



▲ 図 2.9 エラーハンドリング画面

#### 2.4.8 終わりに

感想としては、Next.js 13 の機能は React18 の Suspense や RSC などの機能に合わせたアップデートというのを強く感じました。

# 3

シス研の日常

## 3.1 戦闘機に乗ろう

### 3.1.1 DCS World

#### はじめに

はじめまして、松土です。いきなりですが、僕はミリオタ<sup>\*1</sup>で、特に戦闘機が好きです。僕は戦闘機に乗りたい！！！。しかし、戦闘機というのは皆さんもご存知の通り、軍用の航空機のため主に軍隊が保有しており、日本では航空自衛隊のみが保有しています。そんな戦闘機に乗りようと思ったら、航空自衛隊に入隊し、高い倍率の選抜を受けた後に何年もの厳しい訓練を乗り越えないといけません。そこで、一度は耳にしたことであろう、フライトシミュレーターというのがこの世には存在しており、これは一般人が仮想のコクピットに乗り込み、操縦をシミュレートできる素晴らしいものです。本物のパイロットが実機を操縦する前の訓練をするにあたって使用する事もあります。僕はこれをを利用して戦闘機に乗りたい欲を解消することにしました。

#### DCS World とは

いきなり出てきた DCS World<sup>\*2</sup>とは何か？、これは戦闘機に特化したフライトシミュレーターで、フランス空軍にも採用されている大変優れたものです。現実に存在する戦闘機がいくつかもジュールとしてあり、一般人が家庭でパソコンの中にインストールするだけで、実機を仮想状で操縦できるようになります。プラットフォームは Microsoft Windows で、2008 年からリリースされており、開発元は本社をロシアのモスクワに置いている Eagle Dynamics 社です。

>>>アプリは無料です<<< ※モジュールは有料です

### 3.1.2 動作環境

#### 最小構成

- OS: 64-bit Windows 10 , DirectX11 (version 2.7 以降は Windows 7 非対応かつ Windows 8 が明記落ち、version 2.5.x までは Windows 7/8 も可)
- CPU: Intel Core i3 2.8 GHz / AMD FX
- RAM: 8 GB (重いミッションをプレイする場合: 16 GB)
- HDD 空き容量: 60 GB (※注: 2022 年夏時点の version2.7.16 では 120GB に増加している)
- ビデオカード: NVIDIA GeForce GTX 760 / AMD R9 280X 以上

\*1 ミリタリーオタクの略 所謂軍事が好きな人

\*2 Digital Combat Simulator World の略

- ユーザー認証用インターネット接続

#### 推奨構成

- OS: 64-bit Windows 10 , DirectX11 (version 2.7 以降は Windows 8 が明記落ち、version 2.5.x までは Windows 8 も推奨内)
- CPU: Core i5 3GHz 以上 / AMD FX 又は Ryzen
- RAM: 16 GB (重いミッションをプレイする場合: 32 GB)
- HDD 空き容量: 130 GB (SSD 推奨 全モジュール導入には 460GB)
- ビデオカード: NVIDIA GeForce GTX 1070 / AMD Radeon RX VEGA 56 (8GB VRAM) 以上
- ジョイスティック
- ユーザー認証用インターネット接続

#### VR を使用する場合

推奨構成に上書きして

- CPU: Core i5 3GHz 以上 / AMD FX 又は Ryzen
- RAM: 16 GB (重いミッションをプレイする場合: 32 GB)
- HDD 空き容量: 130 GB (SSD 推奨 全モジュール導入には 460GB)
- ビデオカード: NVIDIA GeForce GTX 1080 / AMD Radeon RX VEGA 64 (8GB VRAM) 以上

### 3.1.3 導入

#### 公式版と Steam 版

公式 DCS World のサイトよりダウンロードする場合とゲームプラットフォームで有名な Steam でダウンロードする場合があります。特に両者違いはありませんが、Steam 版は公式版よりも更新が遅いことがあります。

#### 安定版と Open Beta 版

新しいモジュールはまず Open Beta 版のみに対して提供され、何度かの Open Beta アップデートを経て十分にバグを無くしてから安定版への提供となります。

#### Open Beta の利点

マルチプレイサーバーの多くは Open Beta を使用している (2023/05/07 現在) ためマルチプレイをしたいのなら Open Beta 推奨新しいモジュールをより早く遊ぶことができるレーダーや FLIR など、前バージョンではモジュールへの実装がオミットされていた一部機能や

兵装が追加実装されたのを早く体験できる

### Open Beta の欠点

Open Beta は当然バグが多く、特定の武器を使うとゲームがクラッシュする現象が起きることもある。ゲームがクラッシュまではしなくとも、以前のバージョンでは正常だったはずの特定のミサイルの誘導能力がおかしくなっている、レーダーや照準などの装置の操作や挙動がおかしくなっている、といったバグが新しく増えていることもある。

### モジュールの購入

アプリ自体は無料のため、インストール後起動し、プレイすることが可能ですが、初期で乗ることのできる機体は、第二次世界大戦で使用されたプロペラ機の練習機版 TF-51 とソ連<sup>\*3</sup>が開発した攻撃機 Su-25 だけです。どちらも、戦闘機ではない上にカッコ悪い<sup>\*4</sup>です。

注意として、機体内部のボタンやスイッチを一つ一つまで操作できる機体（クリッカブル機）と、キーボードを使い、キーで細かい機体の操作を行う機体（FC3 機）があり、クリッカブル機は基本的に高価ですが、実機と同じく全てのボタンが操作でき、リアルな操作を楽しむことができます。

### 3.1.4 まとめ

今回は、ミリオタ向けフライトシミュレーターとして、DCS World を紹介させていただきました。導入した後の楽しみ方は、実機と同じエンジンスタートを行ってみたり、マルチプレイで友人と飛んでみたり、または戦ってみたり、と様々であり、あなたの思うがままにやりたいことができるのがこの DCS World の良いところです。もしも、これを読んでいるあなたが DCS World を始めたら、僕と一緒に飛びましょう。

---

<sup>\*3</sup> ソビエト連邦の略

<sup>\*4</sup> あくまで個人の感想です

# 4

## あとがき 枠

### 4.1 本誌創刊に寄せて

物は試し、ということわざがあります。また、言うは易く行なうは難し、ということわざもあります。どちらのことわざにしても、実際にやってみれば良く分かる、という共通の帰結があります。しかし、あまりに簡単に実行できる仕組みを作ってしまうと、しばしば人間は仕組み無しで試すことや仕組み自体を軽視し始め、その仕組みの存在のありがたさを忘れてしまうものなのでしょう。そして、失って初めてその存在のありがたみに気が付く、という言葉は、前述のような体験を経た（やってみた）人からしばしば生まれてくる言葉なのでしょう。では、あまりに簡単に成功する仕組みがある状況下で、次代にその仕組みの意味や価値を伝えるにはどうすればよいのでしょうか。

次代の人々に対して、比較的短期間かつ効果が期待できる方法のひとつに、意図的な損失を生み出す方法があります。失って気が付くのであれば一度失ってみさせれば良い、という考えです。しかし、一度得たものを失うことに少なからず不満を抱いてしまうのは、人間の性です。故に、実行者にはその不満の矛先が向けられる可能性があります。また、損失は発展を阻害する要因でもあります。従って、進歩のための損失であるように、損失の度合いには十分に配慮しなければなりません。

一方で、正義は常に正しいとは限らない、という考えがあります。それは、時代や地域や環境によって、文化や思想や理念など、是非を判断する上での前提条件が異なるからです。ある正義の下では正当だとされる仕組みでも、別の正義の下では不当だとされ得るということです。

す。しかしながら、正義は人間の行為における動機付けのひとつとされています。昨日までの正義を否定するような今日の正義に直面した時、我々はどのように受け止め考えていいかのでしようか。選択肢として、一切の拒絶をするか、昨日までの自らの行いを否定することによって自己正当化をするか、などが考えられます。それらの中でも、多少なりとも理解を試み受け入れようとする選択が、多角的な視点からより良い考えが期待できるでしょう。

私はシス研に長らく在籍していました。本誌のような試みが10年振りに復活し、製本されることを大変嬉しくありがとうございます。シス研では、ある時は観察者として、またある時は友や助言者として、そして敵として、振舞いました。かつて理想と信念を掲げ、仲間と共にある時代を築いた者のひとりとして、自主的に行ったとはいえ、次代の芽が出るまでの橋渡し役はなかなかに堪えるものでした。自らが信じた正義と次代を担う彼らの正義を常に見比べ、その上で役割を果たす必要があったからです。何かを手に入れようとする若さを生かすためには、何かを失うまいとする老いた私情は捨て去らねばなりません。しかしながら、過去の歴史を知り同じ過ちを繰り返さないようにすることは、未来を思い描くために重要なことであるはずです。ある意味ではそれを正義として、客観的な答えを導くために私は自分自身を自制していたのかもしれません。少なくとも、未来を思い描く先導者にとって、彼ら自身が追従するような相手はいませんから。

これから、シス研は新たな時代を迎えようとしています。最後に、マハトマ・ガンジーが残した2つの名言で締めつつ、シス研の今後の活動にご期待ください。

物事は初めはきまつて少数の人によって、ときにはただ一人で始められるものである。  
満足は努力の中にあって、結果にあるものではない。

## 4.1 奥付け

---

### Sysken の技術本 様々な技術を詰め合わせてみました。

---

発行日	2023 年 5 月 28 日	(初版)
サークル	愛知工業大学 システム工学研究会	
Instagram ID	@ait.sysken	
Twitter ID	@set_official	
QiitaOrganizationURL	<a href="https://qiita.com/organizations/sysken">https://qiita.com/organizations/sysken</a>	
代表	牧野遙斗	
代表者メールアドレス	harutiro2027@icloud.com	
企画・編集	牧野遙斗 (Twitter: @minesu1224)	
著者	林航平 すださんかり hihumikan Beyond Toyama 水谷祐生 (Twitter @l8ZAFNZbON1eDbZ) BlacKnight 松土 (Twitter:@kk22blacknight) shirataki1126	
印刷所	しまや出版	

---

※本書の無断複写、複製、データ配信はかたくお断りいたします。