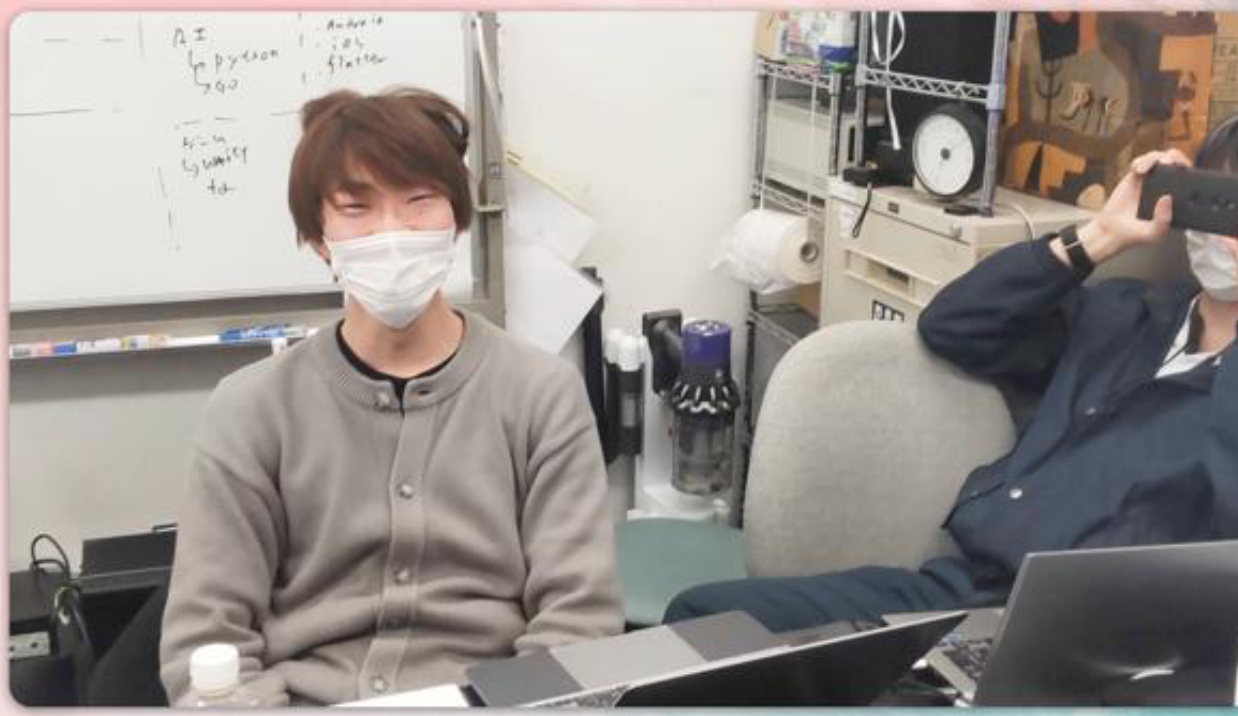
シス研の技術本
テスト作成
表紙



目次

1 これは chapter

1.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

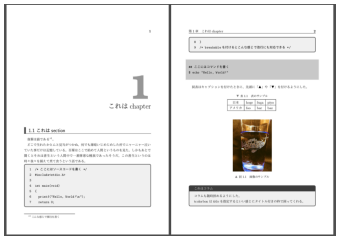
```
6    printf("Hello, World!\n");
7    return 0;
8  }
9  /* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 1.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 1.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。

2

DiscordBot を作ってみよう!

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。[*1] 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。

2.2 実行環境・使用技術

- Python 3.10.8

2.3 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。

2.3.1 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。[*2]

2.3.2 ファイルの作成

Bot を実行する Python ファイルを作ります。

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

[*1] 今回作る DiscordBot のサンプルプログラム https://github.com/sudamichiyo/Discord_Bot_sampleprogram

[*2] Discord の Bot の作り方 (メモ)https://note.com/exteoi/n/nf1c37cb26c41(参照 2023.3.29)

2.3.3 discord.py の準備

ここからは 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。[*3] 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。

```
python -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')
```

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

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

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ところで python3 main.py を実行すると、Bot のサーバが立ち上がります。Bot サーバ起動後に Bot のいるサーバでメッセージを投げると、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。

2.3.4 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。

```
pip 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。

2.3.5 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。

2.4 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。

3
これは chapter

3.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

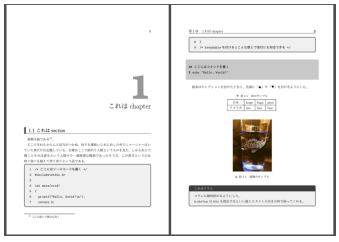
```
6    printf("Hello, World!\n");
7    return 0;
8  }
9  /* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 3.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 3.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。

4

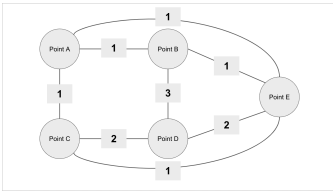
Gin × Neo4j × Docker で最短経路を返してくれる API サーバを建てる

4.1 Gin × Neo4j × Docker で最短経路を返してくれる API サーバを建てる

4.1.1 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。

4.2 今回扱うデータの図



4.3 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
```

4.4 Neo4j に最初にインポートするファイル

point.csv

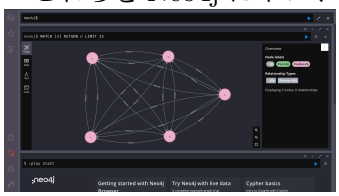
```
1 point_id:ID,point_name,:LABEL
2 a,PointA,Point;Position
3 b,PointB,Point;Position
4 c,PointC,Point;Position
5 d,PointD,Point;Position
6 e,PointE,Point;Position
```

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。



無事できてますね。

4.4.1 Docker 環境の設定

```
1 version: "3"
2 services:
3   go:
4     container_name: NEO4JAPI_GO
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: NEO4JAPI_NEO4J
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

```
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

```
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

```
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"
```

4.5 Go のソースコード

4.5.1 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。

```
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。

4.5.2 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"), neo4j.BasicAu
14  if err != nil {
15  log.Fatal(err)
16  }
17  ses := dr.NewSession(neo4j.SessionConfig{AccessMode: neo4j.AccessM
18  return ses
19 }
20
```

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。

4.5.3 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:Position {point_name
22  path=allShortestPaths ((from)-[distance:Distance*]->(to))
23  WITH
24  [position in nodes(path) | position.point_name] as name,
25  REDUCE(totalMinutes = 0, d in distance | totalMinutes + d.cost) as
要時間
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)
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。

4.5.4 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。

4.5.5 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。

4.5.6 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  }
```

4.6 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。

4.6.1 実行確認

sample.http

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

5 これは chapter

5.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

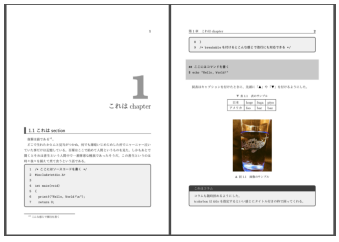
```
6    printf("Hello, World!\n");
7    return 0;
8  }
9  /* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 5.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 5.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。

6
これは chapter

6.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

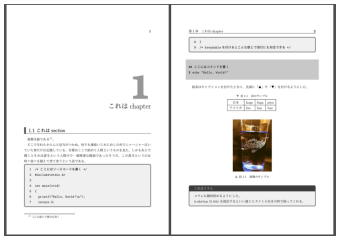
```
6    printf("Hello, World!\n");
7    return 0;
8  }
9  /* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 6.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 6.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。

7

これは chapter

7.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

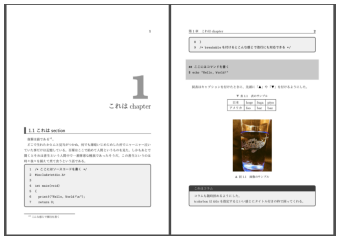
```
6    printf("Hello, World!\n");
7    return 0;
8  }
9  /* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 7.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 7.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。

8 これは chapter

8.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

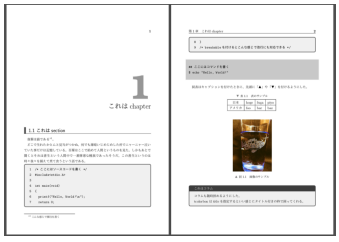
```
6    printf("Hello, World!\n");
7    return 0;
8  }
9  /* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 8.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 8.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。