Dart を用いたプログラミング教育について

箕原辰夫[*1]
Email: *minohara@cuc.ac.jp*

*1: 千葉商科大学政策情報学部政策情報学科

◎Key Words　　Dart, Flutter, Python, アプリケーション開発, プログラミング教育

1. はじめに

Java, JavaScript の後継である Google の Dart プログラミング言語[(1)]は、GUI アプリケーションの開発環境 Flutter[(2)]と共に、これからのクロス・プラットフォームでのアプリケーション開発言語として有望視されている。この Dart と Flutter を用いて、学生が Windows で、iPhone や iPad のアプリケーションを開発できるようにするために、ゼミナールで半期の授業でプログラミング教育を実践した結果について報告する。

2. 授業の構成

ゼミナールの授業週 1 コマ(105 分)を 13 回実施した。受講者は、5 名で、原則対面授業（オンライン・リアルタイムでも受講可能として、その回に出られない学生用に録画も撮った）で、教室にある大型 LED ディスプレイ、および Microsoft Teams の共有画面を提示画面とした。学生には、LED ディスプレイの前面に座って受講してもらうようにした。受講学生は、プログラミングの科目を既修で、一応 Python のプログラムは少しは記述したことがある学生達になっている。ただし、Java または JavaScript を記述したことがない学生が多かった。

表 1 半期 13 回の授業内容

授業回	授業内容
第 1 回	VSCode 上への Dart/Flutter インストール
第 2 回	Dart SDK の設定、それぞれの型のリテラル
第 3 回	変数への代入と型、文字列への式の埋込み
第 4 回	整数除算・剰余、文字列と Unicode の変換
第 5 回	ターミナルからの入力、構造型（レコード型・リスト型・集合型・マップ型)
第 6 回	リストの操作、文字列の分解
第 7 回	format パッケージによる文字列フォーマット、if 文、while 文
第 8 回	if case 文、オプショナル型、for 文、switch 文、try case 文、throw 文、break 文、continue 文
第 9 回	関数の定義、値を返す関数の定義
第 10 回	クラスの定義、Flutter のプロジェクトの作成
第 11 回	継承の定義、Flutter による GUI の作成
第 12 回	Flutter のウィジェットとその配置
第 13 回	Flutter によるアプリケーション作成

3. インストレーション

3.1 VSCode との連携

VSCode[(3)]を使う分には、Dart も Flutter も簡単にインストールできる。と言いたいところだが、それは macOS に限った話である。Flutter のホームページでは、次のように紹介されているが、これは macOS では動くが、Windows では動作しない。

1. VSCode 上で Dart の拡張機能と Flutter の拡張機能をダウンロードし、インストールする。
2. VSCode のコマンドパレットを表示させ、flutter と入力する

以上のような手順で自動的に flutter の SDK がダウンロードされ、インストールできる。ちなみに、macOS の場合は、制約がなかったので、/Applications フォルダの下に flutter をインストールすることにした。Windows では、Program Files フォルダの下に置くことができない。

3.2 ダウンロードベースでのインストール

Windows では、上記の 2.のところでエラーがでるので、次のようにしなければならない。

1. VSCode 上で Dart の拡張機能と Flutter の拡張機能をダウンロードし、インストールする。
2. flutter.dev のページから、Flutter SDK.zip ファイルをダウンロードする。
3. dart.dev のページから、Dart SDK.zip ファイルをダウンロードする。
4. 自分のホームフォルダか、C:ドライブ直下に dev あるいは Developments などの名前でフォルダを作成し、そこに.zip ファイルを展開する。

macOS も Windows も、shell や PowerShell 用に Path の設定をしておく必要がある。この場合、dart だけで開発する場合は、dart-sdk/bin の方を優先させた方が良い。flutter/bin にも dart のコマンドがあるが、こちらは batch ファイルになっている。なお、macOS の場合、上記の手順だけだと flutter の SDK だけがダウンロードされるので、dart.dev からの Dart SDK.zip ファイルをダウンロー

ドして、dart も展開しておいた方が良いと思われる。この場合も、macOS では/Applications フォルダの直下において構わない（dart-sdk というフォルダができる）。

3.3 ターミナル上での実行

PowerShell や zsh などのターミナルからの実行は、実行パスが通っていれば、以下のコマンドで実行することが可能である。dart は、run のオプションで、ほぼインタープリタのように、コンパイルが行なわれることがなく実行が行なわれる。以下の例で、% は shell または PowerShell のプロンプトを示す。

```
% dart run ソースファイル名.dart
```

Flutter の方も、flutter run で実行ができるが、それはプロジェクト中に限られる。flutter の場合は、doctor というオプションがあり、これを使うことで、どのターゲットのプラットフォームで実行ができるかどうかを確かめることができる。

```
% flutter doctor
```

3.4 VSCode での launch.json の設定

VSCode 上では、デバッグの実行までは簡単にできる。まず、.dart の拡張子がついたファイルを作成し、保存すると、ツールバーに▷のデバッグ実行ボタンが表示されるので、それで実行ができる。しかし、これはターミナル上での実行ではないので、「デバッグなしで実行」させることはできない。また、ターミナル上でユーザからの入力があるようなプログラムも実行させることができない。ターミナルから実行させるためには、ソースファイルがあるフォルダの下に.vscode フォルダを作成し、以下のような launch.json を置く必要がある。

```
{
    "version": "0.2.0",
    "configurations": [ {
            "name": "seminar2024",
            "request": "launch",
            "type": "dart",
            "program": "${file}",
            "args": [ "run" ],
            "console": "terminal"
        } ]
}
```

3.5 ライブラリの追加

dart や flutter 上でライブラリを追加するには、以下のようなコマンドをシェル上から発行して、pub.dev[4]というサイトにあるパッケージをダウンロードする必要がある。これで簡単に追加できるので、Python の pip 的な簡単さがある。

```
% dart pub add パッケージ名
% flutter pub add パッケージ名
```

ただし、サードパーティーのパッケージを利用する際には、pubspec.yaml ファイルを、ソースプログラムが置かれているフォルダに記述しておかなければならない。以下は、2024 年 5 月時点での Windows 版のバージョンで記述している。

```
name: アプリケーション名（適当な小文字の名前）
environment:
  sdk: ^3.3.4
```

上記の pub add コマンドを実行すると、これに自動的に dependencies:の行ができて、パッケージの名前とバージョンなどが記述される。

3.6 Web 上での実行

DartPad[5]のページで、Web 上で Dart のプログラムをテキストエリアに貼り付けて実行させることができる。しかしながら、DartPad は、dart:io などの標準ライブラリもサードパーティーのライブラリも利用することができない単なるトイ・インタープリタになっている。

Flutter についても、FlutLab[6]という名前での Web 上で Web アプリケーションを作る開発環境が用意されており、これは VSCode に似た IDE（Integrated Development Environment）である。こちらは充分使い物になるので、Web アプリケーションだけを作る分には、VSCode を使わなくても、これで充分なのではないかと思われる。

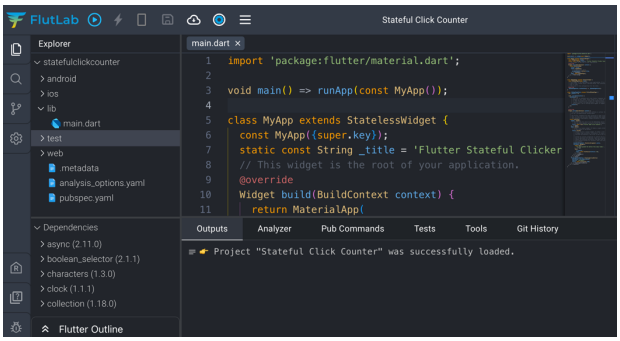

図 1 FlutLab

4. Dart のプログラミング

4.1 式や文の書き方

最初にプログラムの記述感を述べておくと、やはり Java/JavaScript ベースの拡張であるので、Python や Swift と比べると、記述しにくい、記述方法が古い、Java っぽいと感じるところがある。ただ、Rust と比べると、記述感や型推論は優れていると感じられる。プログラムは、C 言語と同様に main 関数の定義から始める。Java からの拡張なので、各文の文末には、;が必要である。

```
void main( ) {
    var   value = 45.3;
    print( "the square of $value is ${value*value}" );
}
```

上記のプログラムをみれば分かるが、Python のフォーマット済み文字列リテラルと同様に、文字列中に「$変数名」あるいは「${式}」という形で変数や式の評価値を文字列に変換して埋め込むことができる。ただし、printf のような表示桁数等の細かなフォーマットを指定するには、サードパーティーのライブラリ (pub.dev に置かれている format パッケージ等) を利用する必要がある。

Swift のオプショナル型と同様に、null 値を取る可能性のある変数や強制的に値を評価するために、?演算子や!演算子が用意されている。JavaScript と異なり、整数除算の演算子が用意されていて、~/で記述する。なお、この頃の言語にありがちだが、整数型は、64bit が標準になっている。その他の演算子や型は、だいたい Java と同様である。ただし、Java と異なり、型推論の機能があるので、変数の宣言では、型名でなくて、JavaScript と同様に var で指定しても構わない。なお、enum 型の拡張である Symbol 型のリテラルも記述できる。「#シンボル名」で記述し、等しいかどうか判定することも可能である。

構造型としては、レコード型、リスト型、集合型、マップ型が用意されている。レコード型は、Python のタプル型のようにも記述できるが、インデックス式やスライス式でアクセスできない。キーワード付きのレコード型のの使い勝手は、C 言語の構造型に似ているが、immutable（書換え不可）なので、あとから属性（フィールド）に値を代入することができない。

4.2　文

if 文に Python の match 文に対応した if case 文がある。ただし、同じ用途で使える switch 文も用意されているので、そちらを使えば事足りてしまう。match 文のガードに対応して、when 句が switch 文の中でも使える。

```
if ( point case [ 0, 0 ] ) { print( "original point" ); }
switch ( point ) {
case [ var x, 0 ] when x >=0: { print( "on plus X-axis" ); }
}
```

if 式については、? : 演算子の他に、オプショナル型に対応した??式が用意されている。??の前の評価値が null の場合は、??の後の値が評価されるという演算子である。

```
int convertToInteger( String? strvalue ) {
  return int.parse( strvalue ?? "0" );
}
```

break 文や continue 文は、C 言語～Java 言語の流れとは異なり、ラベルを取ることはないので、Python と同様に、最深部の繰返しからの脱出・継続しか用意されていない。また、for 文は、C 言語～Java 流の記述以外に、Python 流の for in 文が用意されている。加えて、リスト、集合型、辞書型のデータに対して、Python の forEach( )メソッドが用意されている。以下のように、集合と辞書については、何故かリテラルから直接 forEach が呼べないため、一度変数に代入しておく必要がある。なお、forEach の戻り値は void になっている。

```
for ( var n in [12, 23, 34] ) { print( n ); }
[12, 23, 34].forEach( (n) {print( n ); } );
var aset = {12, 23, 34};
aset.forEach( (n)=> print( n ) );
var adic = {"one":1,"two":2,"three":3};
adic.forEach( (key, value) => print( "$key, $value" ) );
```

4.3　関数の定義

関数の定義の仕方は、ほぼ Java と同じであるが、多相型に対応しており、返す型が決まっていない場合は、以下のように戻り値の型名を省略できる。以下の square は、整数にも実数にも対応した多相型の関数になっている。Rust のように一旦その型の値で呼び出すと、その型専用の関数になるのとは異なり、後から別の型の値でその関数を呼び出すことができる。なお、整数型と実数型の両方で使える num 型も用意されている。

```
square ( var n ) { return n*n; }
```

加えて、Python と同様の無名関数 (lambda 式) を次のような記法で定義することができる。

```
(引数の並び) =>  返す値
```

Python の filter や map、reduce 高階関数に対応するメソッドが、Iterable クラスのオブジェクトにあるので、上記の無名関数を利用して記述することができる。fold は、reduce と似ているが、蓄積値の初期値を指定できる。

```
[12,23,34].where((number) => number.isEven);
[12,23,34].map((number) => number*number);
[12,23,34].reduce( (acc, ele) => acc+ele);
[12,23,34].fold<num>(1, (acc, ele) => acc * ele);
```

Python の影響から、位置引数に加えて、以下のようなキーワード引数、省略可能キーワード引数、省略可能位置引数が用意されているが、Python と異なり、それぞれのタイプの引数の定義の仕方はまったく異なる。なお、それぞれの引数名の後に「=省略時の値」をつけることができる。

```
void displayNumber( {var value} ) {}
void displayNumberOptinal( {num? value} ){}
void displayNumberOptPositional( [num? value ] ){}
```

4.4 クラスの定義

クラスの定義の仕方は、Java とほぼ同じになっている。変更不可能なオブジェクトを生成するためには、const 修飾子がある。また、Python ではなくなった、コンストラクタからオブジェクトを作り出す new 演算子は、省略してもしなくても良くなっている。

コンストラクタは、Java と同様で、クラス名と同じ名前のメソッドを定義する。継承についても、Java と同じで extends とインターフェースを使った implements で指定することができる。クラスメソッドやクラス変数に static を使うのも踏襲されている。public や private 修飾子はなく、Python のように_（アンダーバー）を名前の先頭につけて、クラスの外側から見えなくする。

5. Flutter でのプロジェクト作成

5.1 実はプラットフォーム依存

Flutter のページを見ると一目瞭然だが、開発プラットフォームに応じて、アプリケーションをパブリッシュできるプラットフォームに限りがある。

Windows … Windows, Android, Web
macOS … macOS, iOS, Android, Web

Windows 上において、単体で動くアプリケーションを作成するには、Visual Studio および Visual Studio SDK を必要とする。また、macOS 上において、macOS 用あるいは iOS 用にアプリケーションを作成するには、Xcode が必要になってくる。また、両環境において、Android 用のアプリケーションを作成するには、Android SDK が別途必要になってくる。Web 版（Chrome などで稼働させる）アプリーケーションは、標準で作成することができる。

結局、Windows 上では iOS 用のアプリケーションを作成することができず、作成したい場合には、Dart のソースプログラムを一旦 macOS 上に移して、アプリケーションをビルドする必要がある。ただ、その手間を厭わなければ、Windows 上で iOS 用のアプリケーションも開発することができる。

5.2 Flutter のプロジェクトの作成

VSCode のコマンドパレットを開き、flutter: New Project というコマンドを発行すれば、自動的に flutter が指定したフォルダに新しいプロジェクトを作成してくれる。このフォルダの中の、lib/main.dart が、GUI を作成する Dart のプログラムになっている。このプロジェクトをビルドする（最初にビルドする際に、Web アプリケーションか、それぞれの環境で動作するアプリケーションかを選ぶ必要がある）。学生には、性能が低い Windows PC で、Flutter のプロジェクトをビルドできない場合は、前出の Web 上の FlutLab を使用するように薦めた。

5.3 ウィジェットとその反応

ウィジェットは、自クラスを StatelessWidget あるいは StatefulWidget の子クラスとして定義し、その中で build メソッドを上書き（@override）して配置する。build メソッドでは、レイアウトも含めた各タイプのウィジェットを戻り値として返す形で、配置をする。各ウィジェットを作成する際に、onPressed 等のキーワードパラメータで、マウスが押されたときのコールバック関数（メソッド）を指定するか、Java の AWT の対応するクラスに名前が似ている KeyboardListener クラスの子クラスを定義して、その中の build メソッドで、キー入力を受けるウィジェット配置する。以下は、単なるテキストを表示する例である。

```
import 'package:flutter/material.dart';
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return Center( child: Text( 'Hello, world!',
        textDirection: TextDirection.ltr,
      ) );
  }
}
void main() { runApp( new MyApp() );}
```

6. おわりに

iOS や Android のアプリケーションを作成できるという目的から Dart を選んでみたが、いろいろ紆余曲折はあるものの、Flutter の環境を通して、Windows や macOS の環境かを問わずアプリケーションが作成できるパスができている。以前のクロス・アプリケーション開発環境（Titanium/JavaScript, Xamarin/C#など）は、それぞれの目的の稼働環境にあわせて書き直す必要があったが、Dart はすべて独自の GUI なので、一旦プログラムを記述すれば、書き直しの必要がなく、目的の環境で稼働するようになる。既に Google 系の Web アプリケーションなどでも使われており、今後 Swift と共に普及していくと思われる。プログラムの書き心地は、Java を Python 方向にやや進行させた感じであり、Swift に比べれば、かなり Java 寄りになっているので、今後、Java の後継として、プログラミング教育では使われていくものと思われる。

参考文献

(1) Dart Programming Language, Google, https://dart.dev, (2024 年 6 月閲覧).
(2) Flutter, Google, https://flutter.dev, (2024 年 6 月閲覧) .
(3) VSCode, Microsoft, https://code.visualstudio.com, (2024 年 6 月閲覧).
(4) pub.dev, The official package repository for Dart and Flutter apps., https://pub.dev/, (2024 年 6 月閲覧).
(5) DartPad, https://dartpad.dev/, (2024 年 6 月閲覧).
(6) FlutLab, https://flutlab.io/, (2024 年 6 月閲覧).