2.9 null安全

null許容演算子

Dartではデフォルトが非null許容型です。変数をnull許容型で宣言するときは、型注釈の末尾に?を付与します。null許容型の変数は初期値を省略するとnullで初期化されます。

```
int? num; // nullで初期化される
```

非null許容型の変数をnullで初期化したり、nullを代入することはコンパイルエラーとなります。

```
int nonnullNumber;
nonnullNumber = null; // => A value of type 'Null' can't be assigned to a variabl
e of type 'int'.

int? nullableNumber;
nullableNumber = null; // OK
```

null許容型の変数を扱うには、大きく3つのアプローチがあります。

null認識演算子

?.に続けてプロパティやメソッドを呼び出します。変数がnullの場合はnullが返却されます。

```
String? str;
print(str?.length);
// => null
```

nullアサーション演算子

変数の最後に!を付与することで、非null許容型にキャストできます。ただし、変数がnullの場合は実行時エラーとなります。

```
String? str = // 省略
print(str!.length);
```

タイププロモーション

if文などでnullチェックを行い、変数が必ずnullでないことが明らかな場合は自動的に非null許容型として扱うことができます。

```
String? str;
if (str == null) {
  return;
}
print(str.length);
```

そのほかの便利なnull関連演算子

??オペレータは評価結果がnullだった場合に右辺の値を返します。

```
String? str1 = 'some string';
String? str2 = null;
print(str1 ?? 'null');
// => some string
print(str2 ?? 'null');
// => variable is null.
```

??=オペレータは変数がnullの場合にだけ代入が実行されます。

```
String? str1 = 'some string';
String? str2 = null;

str1 ??= 'new string';
str2 ??= 'new string';

print(str1);
// => some string
print(str2);
// => new string
```

2.10 ライブラリと可視性

Dartでは1つのDartファイルをライブラリと呼びます(**part**命令文により複数のファイルを1つのライブラリとして扱う場合もあります)。外部のライブラリの名前空間にアクセスするには**import**命令を使用します。

```
// Dartの組み込みライブラリは dart: スキームを指定します
import 'dart:math';
// それ以外は package: スキームを指定するのが一般的です
import 'package:path/to/file.dart';
```

Dartには**private**や**public**といった可視性をコントロールするキーワードはありません。デフォルトの振る舞いが**public**に相当し、クラスや関数は**import**命令によってライブラリの外からもアクセスできます。クラス名や関数名を**_**(アンダーバー)ではじめると**private**として扱われ、外部からアクセスできなくなります。

2.11 関数

Dartの関数はトップレベルに定義することができ、Javaなどとは違い必ずしもクラスに属している必要はありません。

Dartの関数を見ていきましょう。

```
String greet(String name) {
  return 'Hello, $name';
}
```

他の多くの言語と記述方法は似ています。この例では、関数**greet()**は**String**型を引数にとり、戻り値が**String**型の関数です。

引数

引数の宣言方法には、省略可能引数と名前付き引数という2つのユニークな仕様があります。

省略可能引数

[ ]で囲った引数は省略して呼び出せるようになります。省略可能引数は省略されるとnullが渡ります。

```
void makeColor(int red, int green, int blue, [int? alpha]) {
  // 省略
}

makeColor(0xFF, 0x00, 0x33); // 引数alphaを省略して呼び出し
makeColor(0xFF, 0x00, 0x33, 0xFF); // 引数alphaを与えて呼び出し
```

省略可能引数はデフォルト値を与えることができます。デフォルト値を与えると、省略可能引数も非null許容型とすることができます。

```
void makeColor(int red, int green, int blue, [int alpha = 0xFF]) {
  // 省略
}

makeColor(0xFF, 0x00, 0x33); // 引数alphaを省略して呼び出し
makeColor(0xFF, 0x00, 0x33, 0x88); // 引数alphaを与えて呼び出し
```

なお、省略可能引数のリストは引数リストの末尾に置く必要があります。

名前付き引数

名前付き引数とは、関数呼び出し時に引数の名前を指定させるしくみです。名前付き引数は引数リストを中括弧({ })で囲います。

```
void makeColor({int? red, int? green, int? blue}) {
  // 省略
}

// 引数の名前を指定する
makeColor(red: 0xFF, green: 0x00, blue: 0x12);
```

名前付き引数はデフォルトでは省略可能として扱われます。必須にする場合は**required**キーワードを与えます。また、デフォルト値を与えることもできます。

```
void makeColor({required int red, required int green, required int blue, int alph
a = 0xFF}) {
  // 省略
}

makeColor(red: 0x78, green: 0x30, blue: 0xBF);
```

名前付き引数は、引数の順番を変えて呼び出すこともできます。

```
void makeColor({required int red, required int green, required int blue, int alph
a = 0xFF}) {
  // 省略
}
makeColor(blue: 0xBF, green: 0x30, red: 0x78);
```

なお、名前付き引数のリストは引数リストの末尾に置く必要がありますが、呼び出し時は位置引数を後方に置いても問題ありません。

```
// 名前付き引数のリストを末尾に置く
void makeColor(String colorName, {required int red, required int green, required
int blue}) {
  // 省略
}

// 呼び出し時は名前付き引数が先頭でも可
makeColor(red: 0x78, green: 0x30, blue: 0xBF, 'purple');
```

関数の省略記法

関数が1つの式からなる場合はアロー演算子を使用した省略記法が利用できます。

```
// 引数を2倍にして返す関数
int doubleValue(int x) {
  return x * 2;
}

// 上の関数を省略記法で宣言
int doubleValue(int x) => x * 2;
```

第一級関数と匿名関数

Dartは第一級関数をサポートした言語です。関数を変数に代入したり、引数に受け取ったりできます。

```
// 引数を2倍にして返す関数
int doubleValue(int x) {
  return x * 2;
}

// 関数doubleValueを変数fに代入
final int Function(int) f = doubleValue;
```

```
final result = f(8);
print("num: $result");
// => num: 16
```

関数オブジェクトの型は、

戻り値の型 Function(引数リストの型)

と宣言します。もちろん型推論も利用可能です。

また、Dartは名前を持たない匿名関数の機能も利用できます。構文は、

(引数リスト) { 関数の本体 }

のようになります。

先ほどの引数を2倍にして返す関数を匿名関数で記述してみましょう。

```
final int Function(int) f = (x) {
  return x * 2;
};
final result = f(8);
print("num: $result");
// => num: 16
```

Dartの匿名関数はクロージャの性質を持ちます。以下のサンプルは**multiple**関数で、引数をキャプチャしたクロージャを生成しています。

```
Function multiple(int i) {
  return (x) => x * i;
}

final f1 = multiple(3);
final f2 = multiple(7);

print(f1(2));
// => 6
print(f2(6));
// => 42
```

2.12 クラス

Dartはクラスベースのオブジェクト指向言語です。すべてのオブジェクトはクラスのインスタンスであり、null以外のすべてのクラスはObjectクラスのサブクラスです。

クラスの定義はclassキーワードに続けてクラス名を記述します。

```
class Point {
  int x = 0;
  int y = 0;
}
```

Pointという二次元座標を表すクラスを宣言しました。xとyの初期値をコンストラクタから与える方法を見てみましょう。

```
class Point {
  Point(int xPosition, int yPosition) : x = xPosition, y = yPosition;

  int x;
  int y;
}
```

コンストラクタの後ろ、:に続けて初期化リストを記述します。上の例のように、コンストラクタ引数を直接初期値として与える記述には糖衣構文が用意されています。

```
class Point {
  Point(this.x, this.y);

  int x;
  int y;
}
```

また、コンストラクタに本体を持たせることも可能ですが、本体が実行される前に非null許容型のクラス変数は初期化済みである必要があります。下記のPointクラスではコンパイルエラーになります。

```
class Point {
  Point(int xPosition, int yPosition) {
    // => Non-nullable instance field 'x' must be initialized.
    // => Non-nullable instance field 'y' must be initialized.
```

```
    x = xPosition;
    y = yPosition;
  }

  int x;
  int y;
}
```

初期化リストでパラメータのアサーションを記述することができます。

```
class Point {
  Point(this.x, this.y) : assert(x >= 0), assert(y >= 0);
  final int x;
  final int y;
}
```

ゲッタとセッタ

すべてのインスタンス変数は暗黙的にゲッタを持ちます。final修飾子のないインスタンス変数は暗黙的にセッタを持ちます。

また、プロパティのカスタムゲッタ、セッタを提供する機能があります。getおよびsetキーワードを利用します。

```
class User {
  User(this.id, this._password);

  final int id;
  String _password;

  // カスタムゲッタ
  // パスワードを伏せ字にして返す
  String get password => '*******';

  // カスタムセッタ
  // パスワードをハッシュ化して保存する
  set password(String newPassword) {
    _password = hash(newPassword);
  }
}
```

いろいろなコンストラクタ

Dartのクラスのコンストラクタを3つ紹介します。