```
# 指定なし
shared_preferences: any ——❻

# 未指定
shared_preferences: ——❼
```

通常は❶のようなキャレット構文でバージョン番号を指定することが推奨されています。キャレット構文は、「指定したバージョンから、互換性のある最新のバージョンまで」の範囲を意味します。互換性の有無はセマンティックバージョニングというルールで決められており、この❶の例では2.1.0以上、3.0.0未満を意味します。❷〜❹のように等号、不等号を組み合わせてバージョンの範囲を指定することも可能です。

❺以降は推奨されない書き方になります。❺のようにバージョンを固定することもできますが、依存関係の解決が困難になる可能性があります。❻はすべてのバージョンを許可します。❼も同じ意味となります。

最終的に解決されたバージョン番号は`pubspec.lock`に記録されます。チーム開発を行う際には`pubspec.lock`を共有することで、メンバー間で同じバージョンで開発することを保証できます。

パッケージバージョンの更新方法

導入しているパッケージの最新バージョンや、更新の可否は以下のコマンドで確認できます。

```
$ flutter pub outdated
```

出力は**図4.2**のようになります。

> **Tips セマンティックバージョニングについて**
>
> pubはセマンティックバージョニングというルールに従って依存関係を解決します。セマンティックバージョニングは、`major.minor.patch`の3つの数字でバージョンを表現します。互換性のない変更があった際にはmajorバージョンが上がります。ただし、1.0.0未満は特別でいかなる変更も起こり得るとされています。Dartのパッケージにおいては、慣習的に1.0.0未満で互換性のない変更があった場合はminorバージョンを上げることとしています。

図4.2 **flutter pub outdatedの出力例**

```
Showing outdated packages.
[*] indicates versions that are not the latest available.

Package Name              Current   Upgradable  Resolvable  Latest

direct dependencies:
shared_preferences        *2.1.0    2.1.1       2.1.1       2.1.1
```

出力されたバージョン番号は**表4.2**のような意味を表します。

表4.2 **バージョン番号の意味**

バージョン名	解説
Current	pubspec.lockに記述されている現在のバージョン
Upgradable	pubspec.yamlに記述された範囲で解決可能な最新バージョン
Resolvable	pubspec.yamlの制約にかかわらず解決可能な最新バージョン
Latest	最新の安定リリースバージョン

図4.2の出力結果からパッケージshared_preferencesは、pubspec.yamlに記述された範囲で最新の2.1.1に更新できるようです。

「Upgradable」のバージョンへ更新するには以下のコマンドを実行します。

```
$ flutter pub upgrade shared_preferences
```

更新されたパッケージとバージョンはターミナルに出力されます。

```
Resolving dependencies...
> shared_preferences 2.1.1 (was 2.0.0)
```

前項のキャレット構文を利用していれば互換性を保ったまま最新バージョンに更新されますが、念のため関連するコードの動作確認やリリースノートの確認を行いましょう。

「Resolvable」のバージョンへ更新するには--major-versionsオプションを付与します。pubspec.yamlも自動的に更新されますので注意してください。

```
$ flutter pub upgrade --major-versions shared_preferences
```

「Resolvable」のバージョンが「Latest」と異なる場合は、以下のコマンドを実行し依存関係を調査してみましょう。競合する依存関係によって最新のバージョンを利用できない場合があります。

```
$ flutter pub deps
```

以下は出力例です。パッケージの依存関係がツリー状に表示されます。ここから競合する依存関係を探し、利用可能なバージョンを調整します。

```
Dart SDK 2.19.5
Flutter SDK 3.7.8
flutter_playground 1.0.0+1
└── shared_preferences 2.1.1
    ├── flutter...
    ├── shared_preferences_android 2.1.4
    │   ├── flutter...
    │   └── shared_preferences_platform_interface...
    ├── shared_preferences_foundation 2.2.2
    │   ├── flutter...
    │   └── shared_preferences_platform_interface...
    └── shared_preferences_platform_interface 2.2.0
        ├── flutter...
        └── plugin_platform_interface 2.1.4
            └── meta...
```

4.2 アプリを日本語に対応させる

Flutterは他の多くのUIフレームワークと同様に国際化するための機能が提供されています。「私の開発するアプリは日本語のみをサポートするから関係ないや」と思った読者の方はお待ちください。Flutterはデフォルト言語が英語で開発されているため、特に対応しなければ意図せず英語が表示されたり、英語圏の日付フォーマットが適用されたりします。そのため、日本向けのアプリを開発するためには、日本向けのローカライズが必要なのです。本章では意図せず英語表示されるケースの紹介と、日本向けのローカライズ方法などを紹介します。

Dartのソースコード以外に、プロジェクトの設定ファイルを変更する必要があります。本節の解説は新しいプロジェクトを作成し、そのプロジェクトに変更を加えながら進めていきます。プロジェクトの作成方法は第1章を参照してください。

意図せず英語表示されるケース

それでは意図せず英語表示されるケースについて具体例を見ていきましょう。

フレームワークが提供する表示文字列

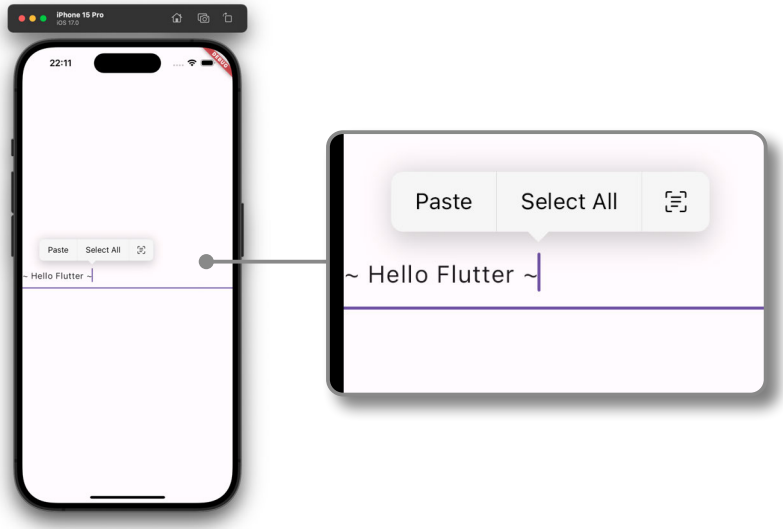
ウィジェットの中には、Flutterフレームワークから表示文字列を提供するものがあります。Flutterはデフォルト言語が英語で開発されているため、特別に対応しなければ英語でしか表示されません。例を見てみましょう。

./lib/main.dart

```
import 'package:flutter/material.dart';

void main() {
  runApp(
    /* ◆ MaterialApp
    マテリアルデザインに準拠したテーマの提供や
    画面遷移の機能を内包したWidget */
    const MaterialApp(
      home: HomeScreen(),
    ),
  );
}

class HomeScreen extends StatelessWidget {
  const HomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    /* ◆ Scaffold
    マテリアルデザインの基本的なレイアウトを提供するWidget */
    return Scaffold(
      body: Center(
        child: Column(
          // 子Widgetを中央へ寄せるためのパラメータ
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            /* ◆ TextField
            ユーザーがテキスト入力するウィジェット */
            const TextField(),
          ],
        ),
      ),
    );
  }
}
```

アプリを実行し、テキストフィールドに任意の文字を入力後、テキストフィールドをロングタップしてみましょう。すると、コンテキストメニューが表示されます(**図4.3**)。

図4.3　**英語のコンテキストメニューが表示される様子**



コンテキストメニューは「Paste」や「Select All」などの項目が表示されます。これらの項目は、ローカライズする実装をしていないので、端末の言語設定を変更しても日本語に切り替わりません。

日付フォーマット

続いて、現在日付を文字で表示する実装を試してみます。日時情報を扱う`DateTime`クラス、日付フォーマットを扱う`DateFormat`クラスを使います。

先ほどのサンプルに追加する形で実装します。`DateFormat`はintlパッケージに含まれていますので、まずはこのパッケージを導入します。プロジェクトのディレクトリで、ターミナルから以下のコマンドを実行してください。

```
# intlパッケージを導入
$ flutter pub add intl:any
```

ここではバージョンを指定せずに`any`を指定しています。通常、`any`でのバージョン指定は推奨されませんが、intlに関しては後述のflutter_localizationsパッケージと併用し、flutter_localizationsに依存したバージョンを採用する

ため、anyを指定することが一般的です。この後flutter_localizationsを導入するので、ここではanyを指定しておきます。

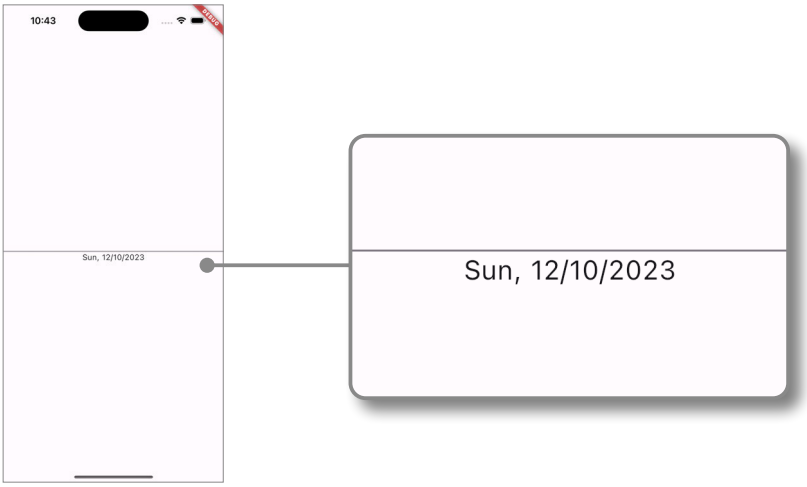
続いて、DateFormatクラスを使って現在日付をフォーマットした文字列を表示するコードを追加します(❶)。

./lib/main.dart

```
import 'package:flutter/material.dart';
import 'package:intl/intl.dart'; // intlパッケージをインポート

void main() {
  runApp(
    const MaterialApp(
      home: HomeScreen(),
    ),
  );
}

class HomeScreen extends StatelessWidget {
  const HomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const TextField(),
            Text(
              DateFormat.yMEd().format(DateTime.now()),  ❶
            ),
          ],
        ),
      ),
    );
  }
}
```

すると表示結果は**図4.4**のようになります。

図4.4 フォーマットされた日付が表示される様子



フォーマットされた日付文字列が表示されました。ただ、日本語圏では年→月→日の順で表示するのが一般的ですが、この表示は英語圏で一般的な月→日→年の順で表示されています。また、曜日も英語表記になっています。

アプリを日本にローカライズする

それではFlutterアプリを日本語化する手順を紹介します。

フレームワークが提供する表示文字列を日本語化する

フレームワークが提供する表示文字列はデフォルトでは英語のみでした。これらの翻訳情報はflutter_localizationsパッケージとして提供されています。さっそく、パッケージを導入してみましょう。プロジェクトのディレクトリで、ターミナルから以下のコマンドを実行してください。

```
# flutter_localizationsパッケージを導入
$ flutter pub add flutter_localizations --sdk=flutter
```

続いて、MaterialAppウィジェットにいくつかパラメータを付与します。

./lib/main.dart
```
import 'package:flutter/material.dart';
import 'package:flutter_localizations/flutter_localizations.dart'; // flutter_loc
alizationsパッケージをインポート
```

```
import 'package:intl/intl.dart';

void main() {
  runApp(
    const MaterialApp(
      localizationsDelegates: [
        GlobalWidgetsLocalizations.delegate,
        GlobalMaterialLocalizations.delegate,
        GlobalCupertinoLocalizations.delegate,
      ],
      supportedLocales: [
        Locale('ja', 'JP'),
      ],
      home: HomeScreen(),
    ),
  );
}
```

❶
❷

`localizationsDelegates`パラメータへ、表示文字列の翻訳情報などを持っているdelegateを渡します(❶)。マテリアルデザインに準拠したウィジェットで扱う翻訳情報は、`GlobalMaterialLocalizations.delegate`に含まれています。iOSスタイルのウィジェットで扱う翻訳情報は、`GlobalCupertinoLocalizations.delegate`に含まれています。また、`GlobalWidgetsLocalizations.delegate`は、テキストの方向(左から右、または右から左)を扱います。これらのdelegateは、flutter_localizationsパッケージが提供しているものです。

`supportedLocales`パラメータは、アプリがサポートするロケールを渡します(❷)。`Locale`クラスのパラメータは、言語と地域でIANA(*Internet Assigned Numbers Authority*)言語サブタグレジストリに準拠しています。今回は日本語のみをサポートするので、`ja`と`JP`を渡します。

それではアプリを実行してみましょう(**図4.5**)。