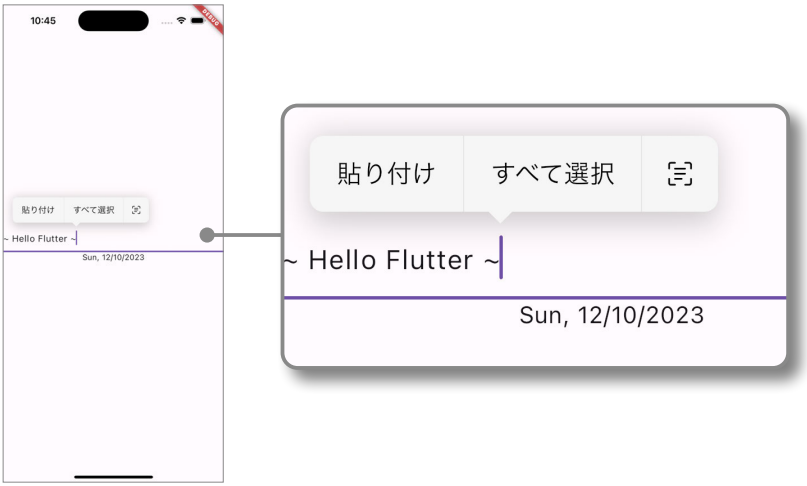
図4.5 **コンテキストメニューが日本語になった様子**




テキストフィールドのコンテキストメニューは日本語になりました。しかし、現在日時の表記は英語のままです。

### 日付フォーマットを日本語化する

`DateFormat`などintlパッケージのAPIは、独自のデフォルトロケールにしたがって動作します。このデフォルトロケールは、`Intl.defaultLocale`で取得や設定ができます。

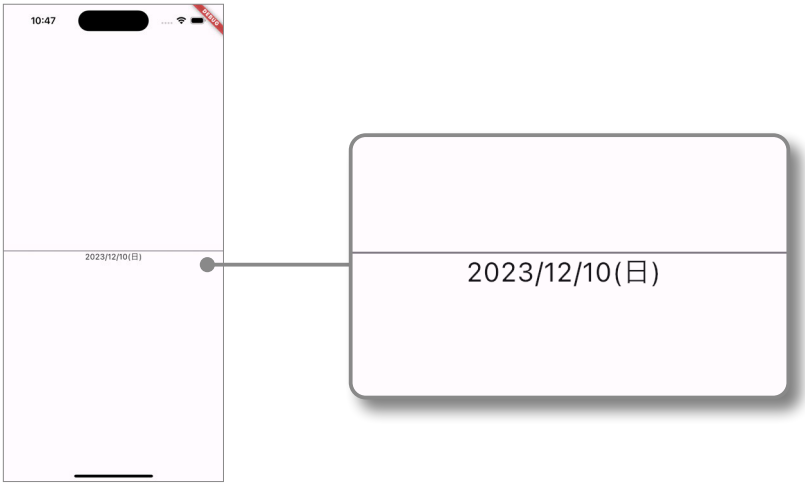
```
./lib/main.dart
class HomeScreen extends StatelessWidget {
  const HomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    // Intl.defaultLocale = 'ja'; ——❶
    Intl.defaultLocale = Localizations.localeOf(context).toString(); ——❷
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const TextField(),
            Text(
              DateFormat.yMEd().format(DateTime.now()),
            ),
```

```
        ],
      ),
    ),
  );
 }
}
```

❶のように、デフォルトロケールに固定値を設定することも可能ですが、のちに複数の言語に対応することになったときのために❷のように設定することもできます。❷はLocalizationsクラスからロケールを取得してデフォルトローケルを設定しています。Localizationsクラスから取得できるロケールは、端末の言語設定をベースにMaterialAppウィジェットのsupportedLocalesパラメータで渡したロケールの中から最適なロケールが選択されます。今回のケースでは、supportedLocalesにLocale('ja', 'JP')のみ設定しているので、端末の言語設定がなんであれja_JPが返却されます。

デフォルトローケルを設定したので、再度アプリを実行してみましょう(**図4.6**)。

図4.6　**日付の表記が日本語になった様子**




現在日時の表記が日本語になりました。これでアプリを日本語化する対応は完了ですが、iOSアプリをアプリストアに公開する際はもう一つ対応が必要です。

#### iOSアプリの対応言語を設定する

次に設定する内容は、主にアプリストア(App Store)に表示されるアプリの対応言語に影響します。修正するファイルは、ios/Runner/Info.plistです。

Info.plistはアプリの構成情報を記述するXML形式のファイルです。CFBundleLocalizationsキーにサポートする言語を記述します。

./ios/Runner/Info.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <!-- 省略 -->
    <key>CFBundleLocalizations</key>
    <array>
        <string>ja</string>
    </array>
</dict>
</plist>
```

今回は日本語のみをサポートするので、jaを設定します。このキーには、MaterialAppウィジェットのsupportedLocalesパラメータで渡したロケールと同じ言語を記述することが、Flutterとして推奨されています。

## メッセージをローカライズする

続いて、アプリ内で表示する独自の文字列(本項では「メッセージ」と呼ぶことにします)をローカライズする方法を紹介します。筆者は日本語のみをサポートするアプリの場合も、ここで紹介する方法を使ってメッセージを扱うことがあります。メッセージの管理がしやすいことや、将来的に複数言語に対応する際にコストが下がるからです。

メッセージのローカライズについてもintlパッケージが提供しています。arbファイルというJSON形式のファイルにメッセージを記述し、コードジェネレータを使って、Dartのコードに変換します。さっそくやってみましょう。

#### コードジェネレータを設定する

pubspec.yamlを編集し、コードジェネレータを有効にします。

```
./pubspec.yaml
flutter:
  generate: true ——❶
```

コードジェネレータの設定は、flutterセクションに記述します(❶)。生成されたコードをプロジェクトから参照できるように以下のコマンドを実行します。

```
$ flutter pub get
```

続いて、ローカライズの構成ファイルを作成します。プロジェクトルートにl10n.yamlというファイルを作成し、以下の内容を記述します。「l10n」は「Localization」の略です(はじまりが「L」、終わりが「n」で、その間が10文字あるので「l10n」)。

```
./l10n.yaml
template-arb-file: app_ja.arb ——❶
output-class: L10n ——❷
nullable-getter: false ——❸
```

❶はarbファイルのテンプレートファイルです。後述の属性を記述するファイルを指定します。

❷はコードジェネレータが生成するローカライズクラスのクラス名を指定します。デフォルトは「AppLocalizations」ですが、筆者はタイプ数を減らすために「L10n」に変更することが多いです。

❸は、ローカライズクラスのゲッタがnull許容型かどうかを指定します。Flutterの後方互換性のためにデフォルトはnull許容型となっていますので、可能であればfalseにしておきましょう。

#### arbファイルを作成する

続いて、lib/l10nディレクトリを作成し、その中にapp_ja.arbというファイルを作成します。

```
./lib/l10n/app_ja.arb
{
  "helloWorld": "こんにちは世界！",
  "@helloWorld": {
    "description": "お決まりの挨拶"
  }
}
```

キーhelloWorldに対して、日本語のメッセージを記述しました。キーの先頭に@を付けると、そのキーは属性を記述するキーとして扱われます。helloWorldのdescription属性(説明)として"お決まりの挨拶"と記述しておきました。

そして次のコマンドを実行します。

```
$ flutter gen-l10n
```

すると、.dart_tool/flutter_gen/gen_l10nディレクトリにarbファイルをもとに生成されたDartファイル(app_localizations.dart)が出力されます。このファイルにメッセージを扱うためのコードが記述されています。次項でこのファイルをインポートしてメッセージを表示するコードを書いてみましょう。

#### ローカライズされたメッセージを表示する

それではメッセージを表示するコードを書いてみましょう。

./lib/main.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart'; ——❶
// import 'package:flutter_localizations/flutter_localizations.dart';
import 'package:intl/intl.dart';

void main() {
  runApp(
    const MaterialApp(
      /*
      localizationsDelegates: [
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
        GlobalCupertinoLocalizations.delegate,
      ],
      */
      localizationsDelegates: L10n.localizationsDelegates, ——❷
      /*
      supportedLocales: [
        Locale('ja'),
      ],
      */
      supportedLocales: L10n.supportedLocales, ——❸
      home: HomeScreen(),
    ),
  );
```

```
}

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

  @override
  Widget build(BuildContext context) {
    Intl.defaultLocale = Localizations.localeOf(context).toString();
    final l10n = L10n.of(context); ——❹
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const TextField(),
            Text(
              DateFormat.yMEd().format(DateTime.now()),
            ),
            Text(l10n.helloWorld), ——❺
          ],
        ),
      ),
    );
  }
}
```

まず生成されたコードをインポートします(❶)。次に`MaterialApp`のパラメータを調整します。`localizationsDelegates`と`supportedLocales`を、コード生成されたものに置き換えます(❷、❸)。

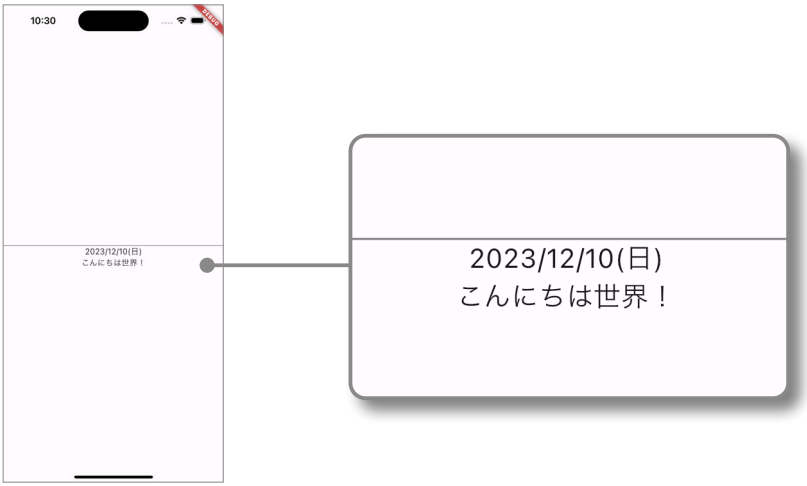
`L10n.localizationsDelegates`はflutter_localizationsが提供するdelegateに加え、❹で取得しているローカライズファイルをロードするdelegateも含まれています。`L10n.supportedLocales`はarbファイルが用意された言語が格納されています。

続いて、生成されたローカライズクラスを取得します(❹)。クラス名は`l10n.yaml`ファイルの`output-class`で指定した`L10n`です。

このクラスはarbファイルのキーに対応するプロパティを持ちます。先ほど作成したarbファイルには`helloWorld`というキーを記述しましたので、`L10n`クラスは`helloWorld`というプロパティを持ちます。それを取り出して`Text`ウィジェットに渡しているのが❺です。

エディタで`helloWorld`プロパティにカーソルを合わせると、arbファイルの`description`属性の値が表示されます(**図4.7**)。

図4.7 メッセージが表示される様子




以上でローカライズされたメッセージの設定が完了しました。

## arbファイルの扱い方

そのほか便利なarbファイルの記述方法や、複数の言語に対応する方法を紹介します。

### プレースホルダ

arbファイルにはプレースホルダという機能があります。例として検索機能のあるアプリを考えてみましょう。検索結果の件数を表示するメッセージを表示するとします。

```
./lib/l10n/app_ja.arb
{
  // 省略
  "numOfSearchResult": "検索結果は{count}件です。",
  "@numOfSearchResult": {
    "description": "検索結果"
  }
}
```

{count}の部分がプレースホルダです。キーnumOfSearchResultに対応するメソッドが生成され、引数としてcountを受け取ります。動作を確認する際

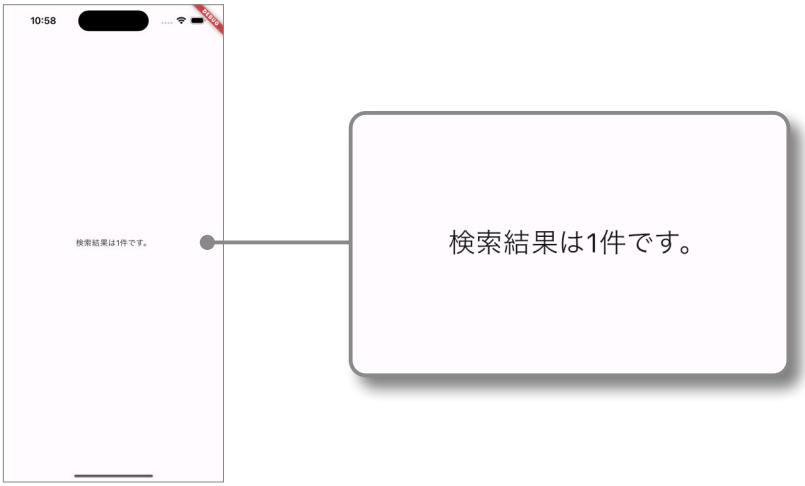
は、以下のコマンドを実行してローカライズクラスを再生成してください。

```
$ flutter gen-l10n
```

```
Text(l10n.numOfSearchResult(1)),
// => 検索結果は1件です。
```

プレースホルダは属性で型を指定したほうが安全です(**図4.8**)。

図4.8 **プレースホルダを利用してメッセージをした様子**




./lib/l10n/app_ja.arb

```
{
  // 省略
  "numOfSearchResult": "検索結果は{count}件です。",
  "@numOfSearchResult": {
    "description": "検索結果",
    "placeholders": {
      "count": {
        "type": "int"
      }
    }
  }
}
```

**単数形と複数形の対応**

日本語ではあまり意識しませんが、英語のように単数形と複数形でメッセージを変える言語もあります。これを実現するための機能があります。