### 1. \*\*Компонент Navigator\*\*: основные методы взаимодействия

`Navigator` в Flutter управляет стеком навигации между разными экранами (или страницами) в приложении. Основные методы для взаимодействия с `Navigator`:

- \*\*`push()`\*\*: Добавляет новый маршрут (страницу) в стек навигации.

```dart

Navigator.push(

context,

MaterialPageRoute(builder: (context) => NewPage()),

);

```

- \*\*`pop()`\*\*: Удаляет верхний маршрут из стека навигации (возвращает пользователя на предыдущий экран).

```dart

Navigator.pop(context);

```

- \*\*`pushReplacement()`\*\*: Заменяет текущий маршрут на новый, удаляя старый из стека.

```dart

Navigator.pushReplacement(

context,

MaterialPageRoute(builder: (context) => NewPage()),

);

```

- \*\*`pushNamed()`\*\*: Осуществляет переход по именованному маршруту.

```dart

Navigator.pushNamed(context, '/home');

```

- \*\*`popUntil()`\*\*: Удаляет все маршруты из стека до определённого, который удовлетворяет условию.

```dart

Navigator.popUntil(context, ModalRoute.withName('/home'));

```

- \*\*`canPop()`\*\*: Проверяет, можно ли вернуться на предыдущий экран (есть ли маршруты для `pop()`).

```dart

if (Navigator.canPop(context)) {

Navigator.pop(context);

}

```

### 2. \*\*Виды `Route`\*\*

`Route` — это объект, который управляет навигацией на определенный экран (страницу). Основные виды маршрутов (routes) в Flutter:

- \*\*`MaterialPageRoute`\*\*: Предоставляет типичную анимацию для платформы Android (переход с помощью слайдов).

```dart

Navigator.push(

context,

MaterialPageRoute(builder: (context) => NewPage()),

);

```

- \*\*`CupertinoPageRoute`\*\*: Предоставляет анимацию переходов, характерную для iOS (скольжение с правой стороны).

```dart

Navigator.push(

context,

CupertinoPageRoute(builder: (context) => NewPage()),

);

```

- \*\*`PageRouteBuilder`\*\*: Позволяет настраивать анимацию переходов вручную. Это полезно, если требуется создать кастомную анимацию перехода между страницами.

```dart

Navigator.push(

context,

PageRouteBuilder(

pageBuilder: (context, animation, secondaryAnimation) => NewPage(),

transitionsBuilder: (context, animation, secondaryAnimation, child) {

var begin = Offset(0.0, 1.0);

var end = Offset.zero;

var curve = Curves.ease;

var tween = Tween(begin: begin, end: end).chain(CurveTween(curve: curve));

var offsetAnimation = animation.drive(tween);

return SlideTransition(

position: offsetAnimation,

child: child,

);

},

),

);

```

- \*\*`NamedRoute`\*\*: Используется для навигации с именованными маршрутами (особенно удобно при сложной структуре приложения).

```dart

Navigator.pushNamed(context, '/home');

```

### 3. \*\*Что такое Platform Method Channel?\*\*

`Platform Method Channel` — это механизм взаимодействия между Dart-кодом Flutter и платформо-специфичным кодом на нативных языках (Kotlin/Java для Android и Swift/Objective-C для iOS). Этот механизм позволяет выполнять задачи, которые требуют доступа к платформенным API (например, получение заряда батареи, доступ к датчикам, работе с Bluetooth и т.д.).

- \*\*Основные компоненты\*\*:

- \*\*Dart (Flutter)\*\*: использует `MethodChannel` для отправки запросов на платформу.

- \*\*Native Code (Android/iOS)\*\*: реализует обработчик вызовов от Flutter через `MethodChannel`.

Пример для Flutter:

```dart

static const platform = MethodChannel('com.example.battery');

Future<void> \_getBatteryLevel() async {

String batteryLevel;

try {

final int result = await platform.invokeMethod('getBatteryLevel');

batteryLevel = 'Battery level: $result%.';

} on PlatformException catch (e) {

batteryLevel = "Failed to get battery level: '${e.message}'.";

}

print(batteryLevel);

}

```

Пример для Android (Kotlin):

```kotlin

class MainActivity : FlutterActivity() {

private val CHANNEL = "com.example.battery"

override fun configureFlutterEngine(@NonNull flutterEngine: FlutterEngine) {

MethodChannel(flutterEngine.dartExecutor.binaryMessenger, CHANNEL).setMethodCallHandler { call, result ->

if (call.method == "getBatteryLevel") {

val batteryLevel = getBatteryLevel()

if (batteryLevel != -1) {

result.success(batteryLevel)

} else {

result.error("UNAVAILABLE", "Battery level not available.", null)

}

} else {

result.notImplemented()

}

}

}

private fun getBatteryLevel(): Int {

val batteryIntent = applicationContext.registerReceiver(null, IntentFilter(Intent.ACTION\_BATTERY\_CHANGED))

val level = batteryIntent?.getIntExtra(BatteryManager.EXTRA\_LEVEL, -1) ?: -1

val scale = batteryIntent?.getIntExtra(BatteryManager.EXTRA\_SCALE, -1) ?: -1

return if (level == -1 || scale == -1) -1 else (level \* 100) / scale

}

}

```

### 4. \*\*Как производится валидация вызываемого method channel?\*\*

Валидация вызываемого метода через `MethodChannel` осуществляется несколькими способами:

1. \*\*Проверка доступности метода на нативной стороне\*\*:

- На стороне платформы нативный код должен проверять, что метод вызова существует. Если метод не поддерживается, он возвращает ошибку:

```kotlin

if (call.method == "getBatteryLevel") {

// Реализация метода

} else {

result.notImplemented()

}

```

2. \*\*Проверка входных данных\*\*:

- Перед выполнением метода можно проверить переданные аргументы (если они есть):

```kotlin

if (call.method == "setAlarm") {

val hour = call.argument<Int>("hour")

val minute = call.argument<Int>("minute")

if (hour != null && minute != null) {

// Выполнение метода

} else {

result.error("INVALID\_ARGUMENT", "Invalid hour or minute", null)

}

}

```

3. \*\*Обработка ошибок\*\*:

- При возникновении ошибок на нативной стороне (например, если невозможно выполнить действие), ошибки можно передавать обратно в Dart:

```kotlin

result.error("UNAVAILABLE", "Battery level not available", null)

```

4. \*\*Возвращаемые значения\*\*:

- Метод может возвращать результат выполнения в виде `success` (успешное выполнение) или `error` (ошибка). Со стороны Flutter можно проверять, что данные корректны, и обрабатывать ошибки.

```dart

try {

final int result = await platform.invokeMethod('getBatteryLevel');

print('Battery level: $result');

} on PlatformException catch (e) {

print("Failed to get battery level: '${e.message}'.");

}

```

Таким образом, валидация включает проверку доступности метода, проверку переданных аргументов и корректную обработку возможных ошибок.