Курс Flutter

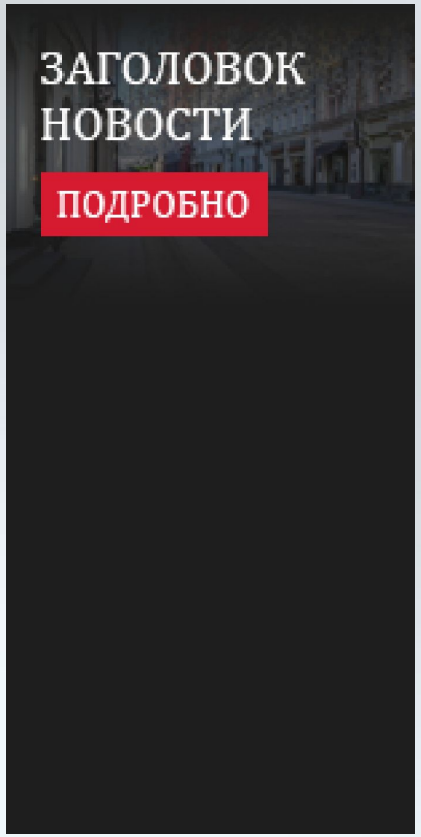
Урок 12. Кросс-платформенные решения

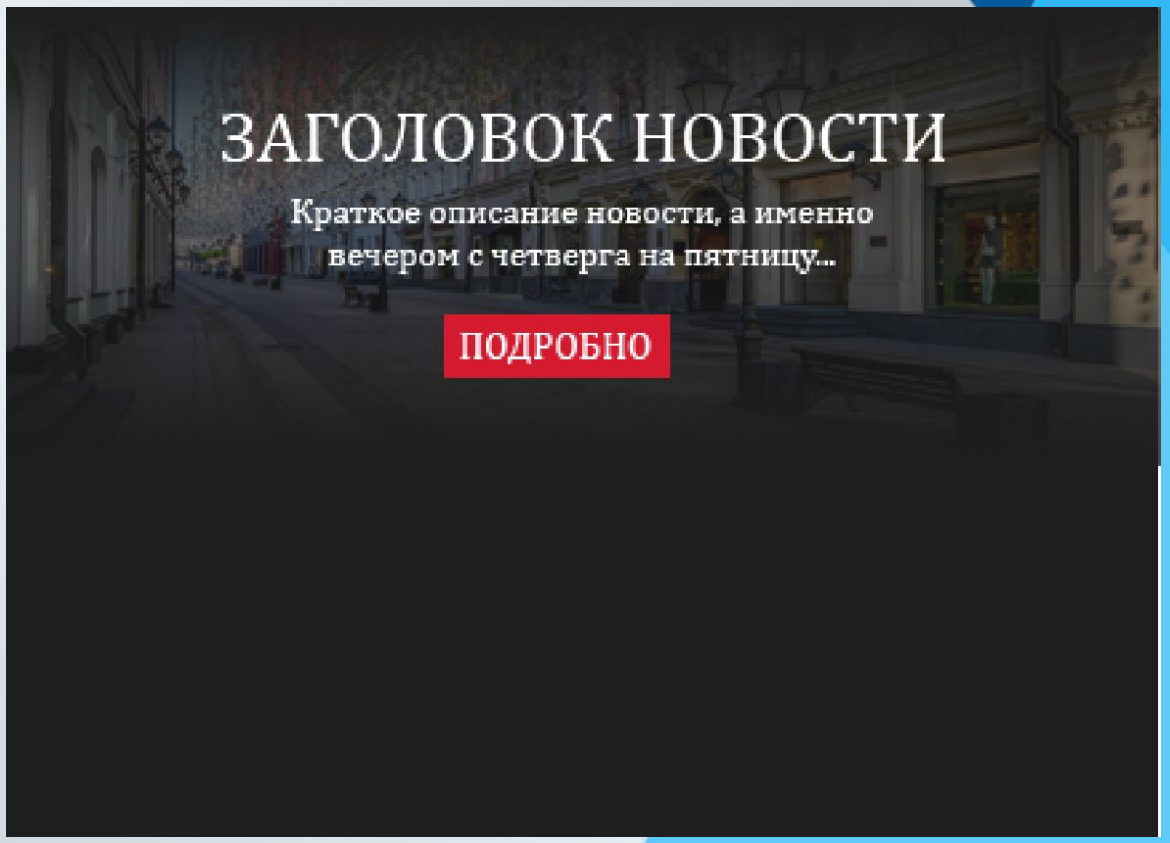
Владимир Полюхович

Урок 12. Кросс-платформенные решения

- Научиться создавать виджеты для адаптивного дизайна
- Научиться работать с виджетом Focus
- Научиться реализовывать собственные политики фокусировки виджетов
- Научиться обрабатывать нажатия кнопок
- Научиться разделять дизайн по платформам
- Научить виджеты взаимодействовать с курсором мыши
- Научиться работать с масштабированием на ТВ

Адаптивность приложения





Адаптивные виджеты

```dart
class MyAdaptivePage extends StatelessWidget {
  const MyAdaptivePage({super.key});

  @override
  Widget build(BuildContext context) {
    return LayoutBuilder(builder: (context, constraints) {
      if (constraints.maxWidth <= 480) {
        return const NarrowPageContent();
      }

      if (constraints.maxWidth <= 600) {
        return const MediumPageContent();
      }

      if (constraints.maxWidth <= 1280) {
        return const LargePageContent();
      }

      throw Error();
    });
  }
}
```

Адаптивные виджеты

```dart
class ResponsiveBuilder<T> extends StatelessWidget {
  final Widget Function(BuildContext, Widget, T) builder;
  final Widget child;
  final T narrow;
  final T large;
  final T medium;

  const ResponsiveBuilder({
    required this.builder,
    required this.narrow,
    required this.large,
    required this.medium,
    required this.child,
    super.key,
  });

  @override
  Widget build(BuildContext context) {
    return LayoutBuilder(builder: (context, constr) {
      if (constr.maxWidth <= ScreenSizes.narrow) {
        return builder(context, child, narrow);
      }

      if (constr.maxWidth <= ScreenSizes.medium) {
        return builder(context, child, medium);
      }

      return builder(context, child, large);
    });
  }
}
```

```dart
class ScreenSizes {
  static const narrow = 480;
  static const medium = 600;
  static const large = 1280;
}
```

Адаптивные виджеты

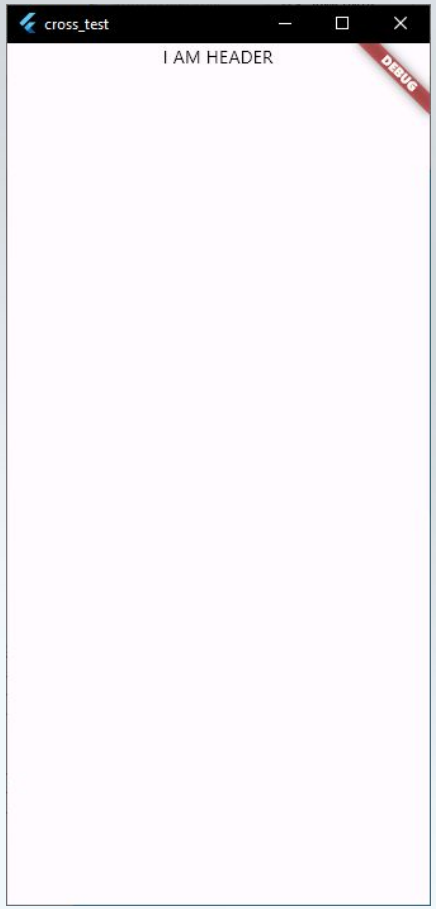
```
class MyAdaptivePage extends StatelessWidget {
  const MyAdaptivePage({super.key});

  @override
  Widget build(BuildContext context) {
    return ResponsiveBuilder<double>(
      narrow: 12,
      medium: 16,
      large: 24,
      builder: (context, child, headerSize) {
        return Text(
          'I AM HEADER',
          style: TextStyle(
            fontSize: headerSize,
          ),
        );
      },
      child: const SizedBox(),
    );
  }
}
```

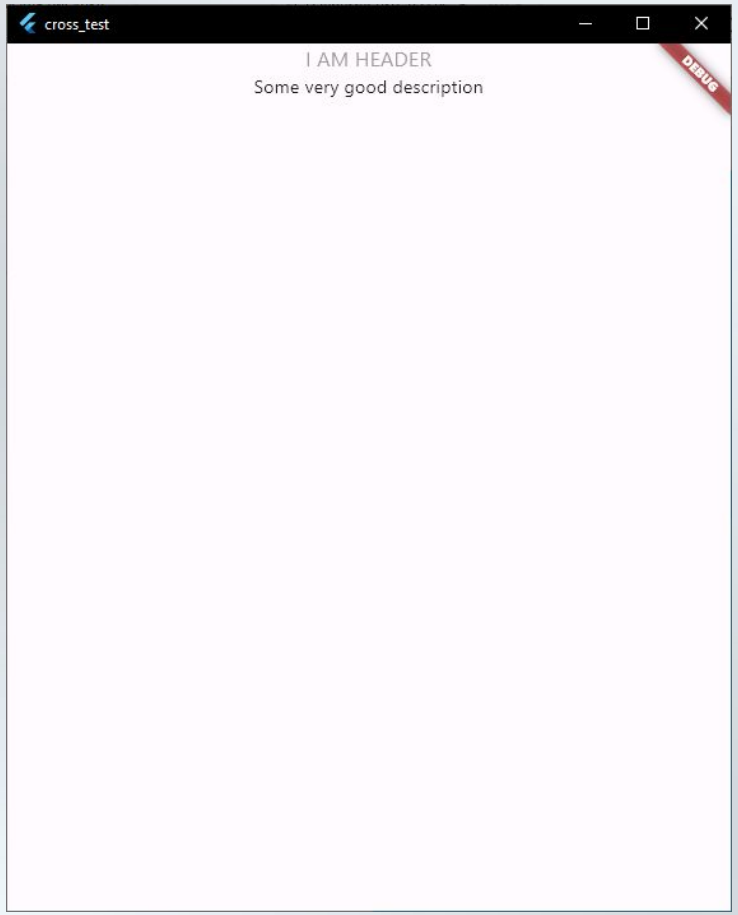
Адаптивные виджеты

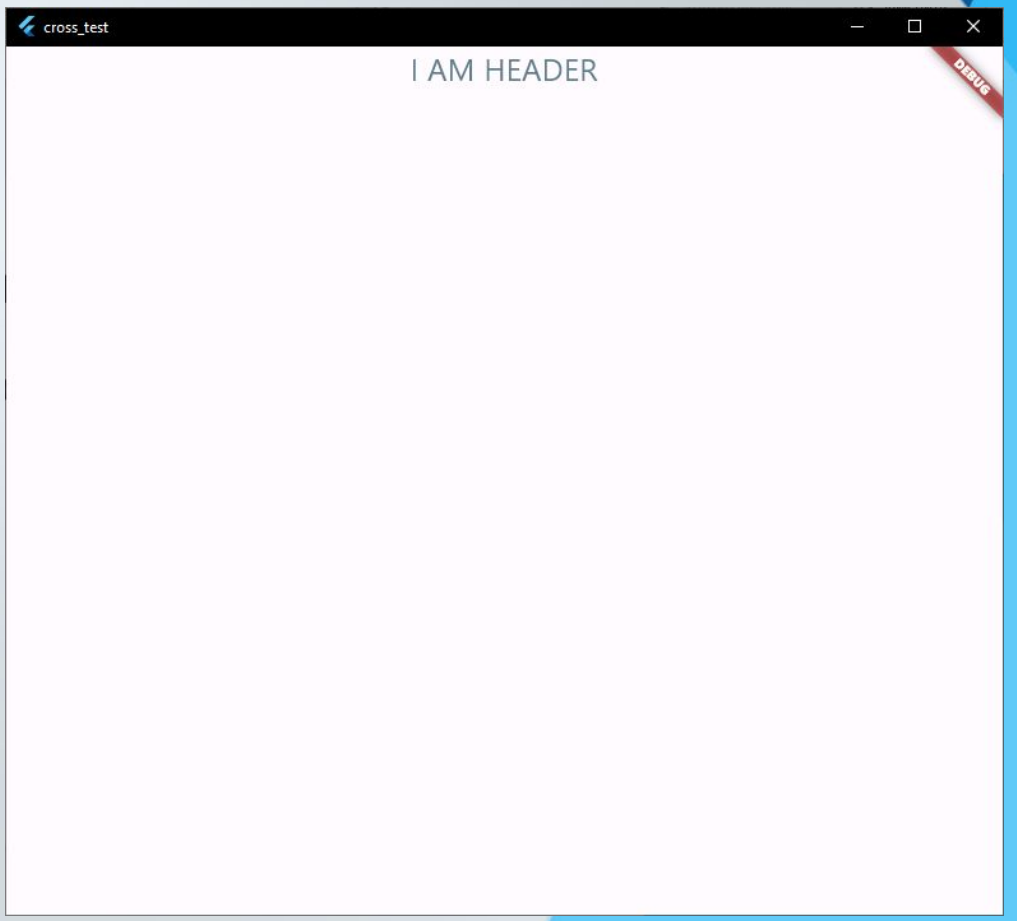
```dart
class MyAdaptivePage extends StatelessWidget {
  const MyAdaptivePage({super.key});

  @override
  Widget build(BuildContext context) {
    return Material(
      child: ResponsiveBuilder<ResponsiveHeaderData>(
        narrow: ResponsiveHeaderData(
          hasDescription: false,
          headerFontSize: 14,
          headerColor: Colors.black,
        ),
        medium: ResponsiveHeaderData(
          hasDescription: true,
          headerFontSize: 16,
          headerColor: Colors.grey,
        ),
        large: ResponsiveHeaderData(
          hasDescription: false,
          headerFontSize: 24,
          headerColor: Colors.blueGrey,
        ),
        builder: (context, child, data) {
          return Column(
            children: [
              Text(
                'I AM HEADER',
                style: TextStyle(
                  fontSize: data.headerFontSize,
                  color: data.headerColor,
                ),
              ),
              if (data.hasDescription) const Text('Some very good description'),
            ],
          );
        },
        child: const SizedBox(),
      ),
    );
  }
}
```

```dart
class ResponsiveHeaderData {
  final double headerFontSize;
  final Color headerColor;
  final bool hasDescription;

  ResponsiveHeaderData({
    required this.headerFontSize,
    required this.headerColor,
    required this.hasDescription,
  });
}
```

Адаптивные виджеты







Адаптивные виджеты

```dart
class ResponsiveWidget extends StatelessWidget {
  final Widget Function(BuildContext) narrow;
  final Widget Function(BuildContext) medium;
  final Widget Function(BuildContext) large;

  const ResponsiveWidget({
    super.key,
    required this.narrow,
    required this.medium,
    required this.large,
  });

  @override
  Widget build(BuildContext context) {
    return LayoutBuilder(builder: (context, constr) {
      if (constr.maxWidth <= ScreenSizes.narrow) {
        return narrow(context);
      }

      if (constr.maxWidth <= ScreenSizes.medium) {
        return medium(context);
      }

      return large(context);
    });
  }
}
```

```dart
class MyAdaptivePage extends StatelessWidget {
  const MyAdaptivePage({super.key});

  @override
  Widget build(BuildContext context) {
    return Material(
        child: ResponsiveWidget(
      narrow: (context) => Column(
        children: items,
      ),
      medium: (context) => Column(
        children: items,
      ),
      large: (context) => Row(
        children: items,
      ),
    ));
  }
}
```

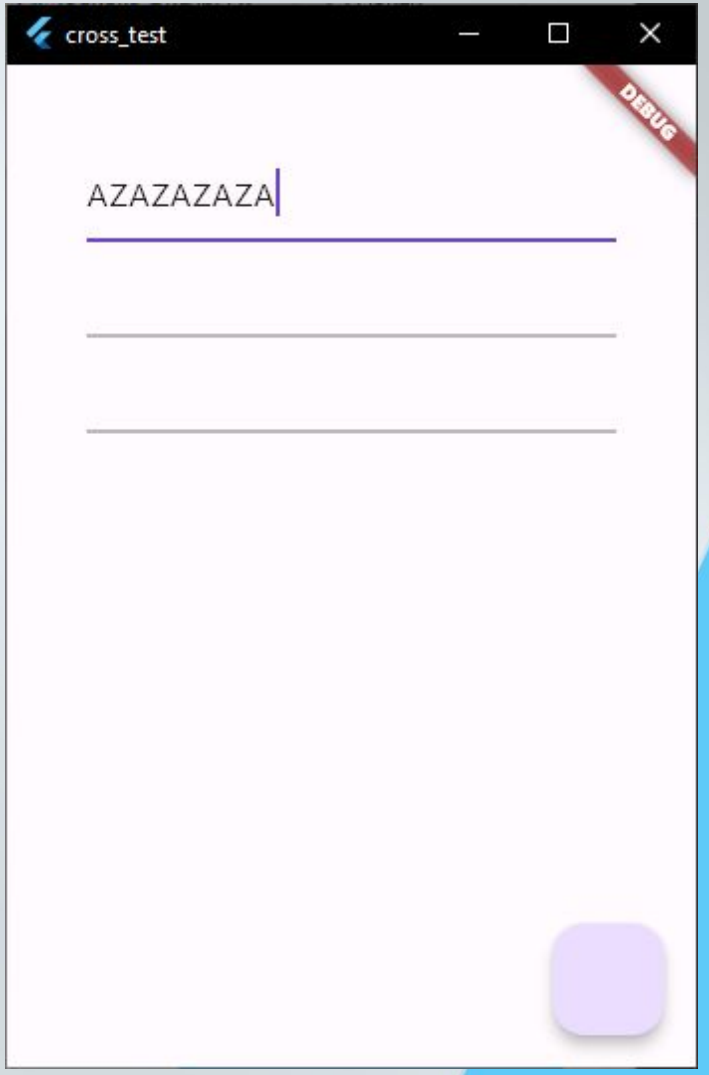
Работа с Focus

Focus – виджет во Flutter, который позволяет сфокусироваться на элементе. Это позволяет нам навигироваться по элементам, используя клавиатуру, а так же фокусироваться на таких виджетах, как TextField и других. У каждого виджета Focus есть своя FocusNode

FocusNode – это класс, предоставляющий данные об фокусируемом элементе. Также, он позволяет программно фокусироваться на элементе, слушать изменения фокуса и др.

Работа с Focus

```dart
class _MyAdaptivePageState extends State<MyAdaptivePage> {
  final _firstItemFocusNode = FocusNode();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      floatingActionButton: FloatingActionButton(onPressed: () {
        _firstItemFocusNode.requestFocus();
      }),
      body: Padding(
        padding: const EdgeInsets.all(40.0),
        child: Column(
          children: [
            TextField(
              focusNode: _firstItemFocusNode,
              autofocus: true,
            ),
            const TextField(),
            const TextField(),
          ],
        ),
      ),
    );
  }
}
```



Работа с Focus

```dart
class FocusTest extends StatelessWidget {
  const FocusTest({super.key});

  @override
  Widget build(BuildContext context) {
    return Material(
      child: ListView(
        children: orders.map((e) {
          return Focus(
            child: Container(
              color: Colors.blueGrey,
              padding: const EdgeInsets.all(16),
              child: Text(
                e.id,
                style: const TextStyle(
                  fontSize: 24,
                  color: Colors.white,
                ),
              ),
            ),
          );
        }).toList(),
      ),
    );
  }
}
```

Работа с Focus

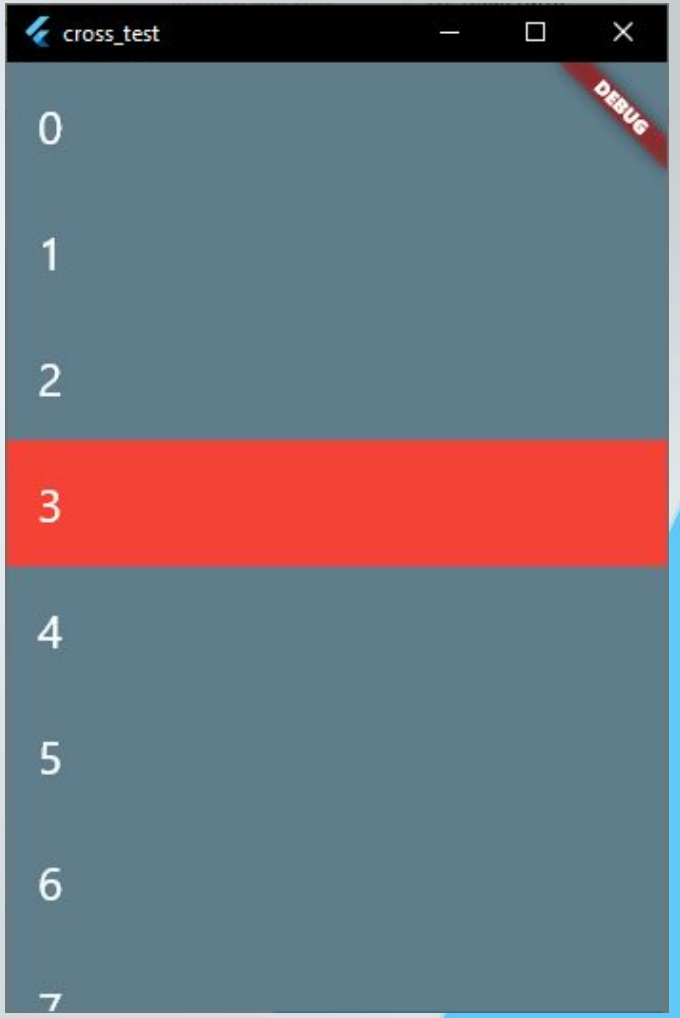
```
class Focusable extends StatefulWidget {
  final Widget Function(BuildContext, Widget, bool) builder;
  final Widget child;
  final bool autofocus;

  const Focusable({
    required this.builder,
    required this.child,
    this.autofocus = false,
    super.key,
  });

  @override
  State<Focusable> createState() => _FocusableState();
}
```

Работа с Focus

```
class _FocusableState extends State<Focusable> {
  final _node = FocusNode();
  var _isFocused = false;

  @override
  void dispose() {
    _node.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Focus(
      autofocus: widget.autofocus,
      onFocusChange: (value) {
        setState(() {
          _isFocused = value;
        });
      },
      child: widget.builder(context, widget.child, _isFocused),
    );
  }
}
```

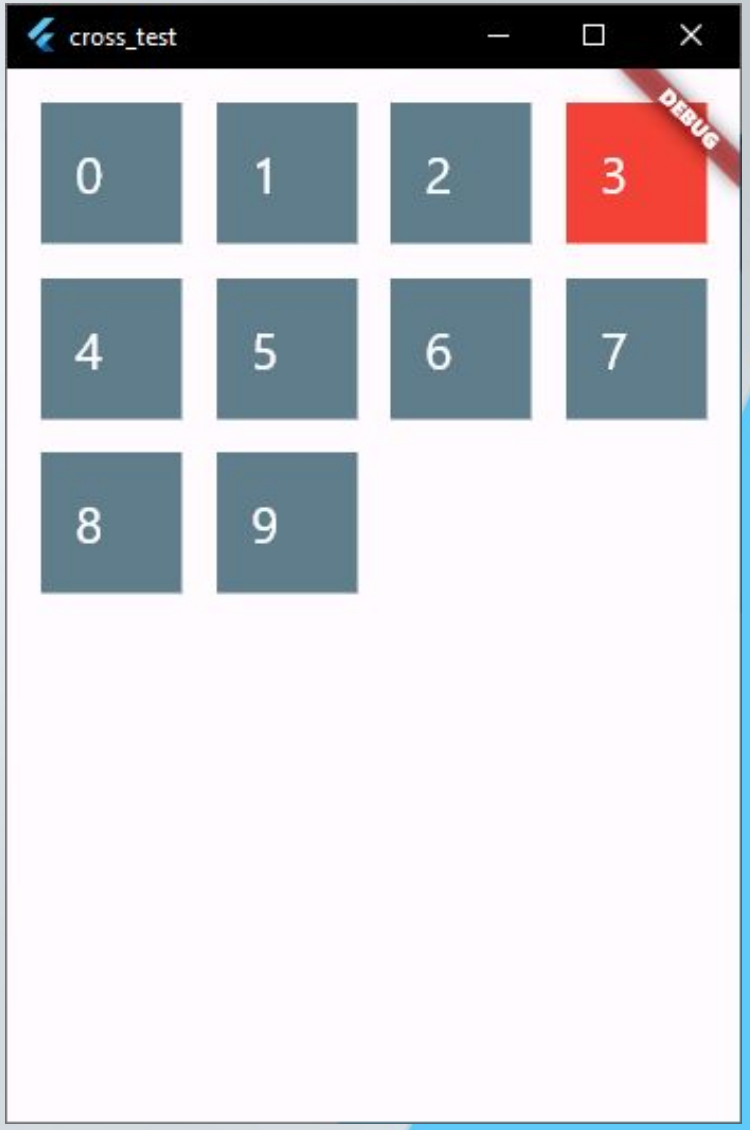
Работа с Focus

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

  @override
  Widget build(BuildContext context) {
    return Material(
      child: ListView(
        children: orders.map((e) {
          return Focusable(
            builder: (context, child, isFocused) => Container(
              color: isFocused ? Colors.red : Colors.blueGrey,
              padding: const EdgeInsets.all(16),
              child: child,
            ),
            autofocus: true,
            child: Text(
              e.id,
              style: const TextStyle(
                fontSize: 24,
                color: Colors.white,
              ),
            ),
          );
        }).toList(),
      ),
    );
  }
}
```



FocusTraversalGroup

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

  @override
  Widget build(BuildContext context) {
    return Material(
      child: Padding(
        padding: const EdgeInsets.all(16),
        child: GridView.count(
          mainAxisSpacing: 16,
          crossAxisSpacing: 16,
          crossAxisCount: 4,
          children: orders.map((e) {
            return Focusable(
              builder: (context, child, isFocused) => Container(
                color: isFocused ? Colors.red : Colors.blueGrey,
                padding: const EdgeInsets.all(16),
                child: child,
              ),
              autofocus: true,
              child: Text(
                e.id,
                style: const TextStyle(
                  fontSize: 24,
                  color: Colors.white,
                ),
              ),
            );
          }).toList(),
        ),
      ),
    );
  }
}
```



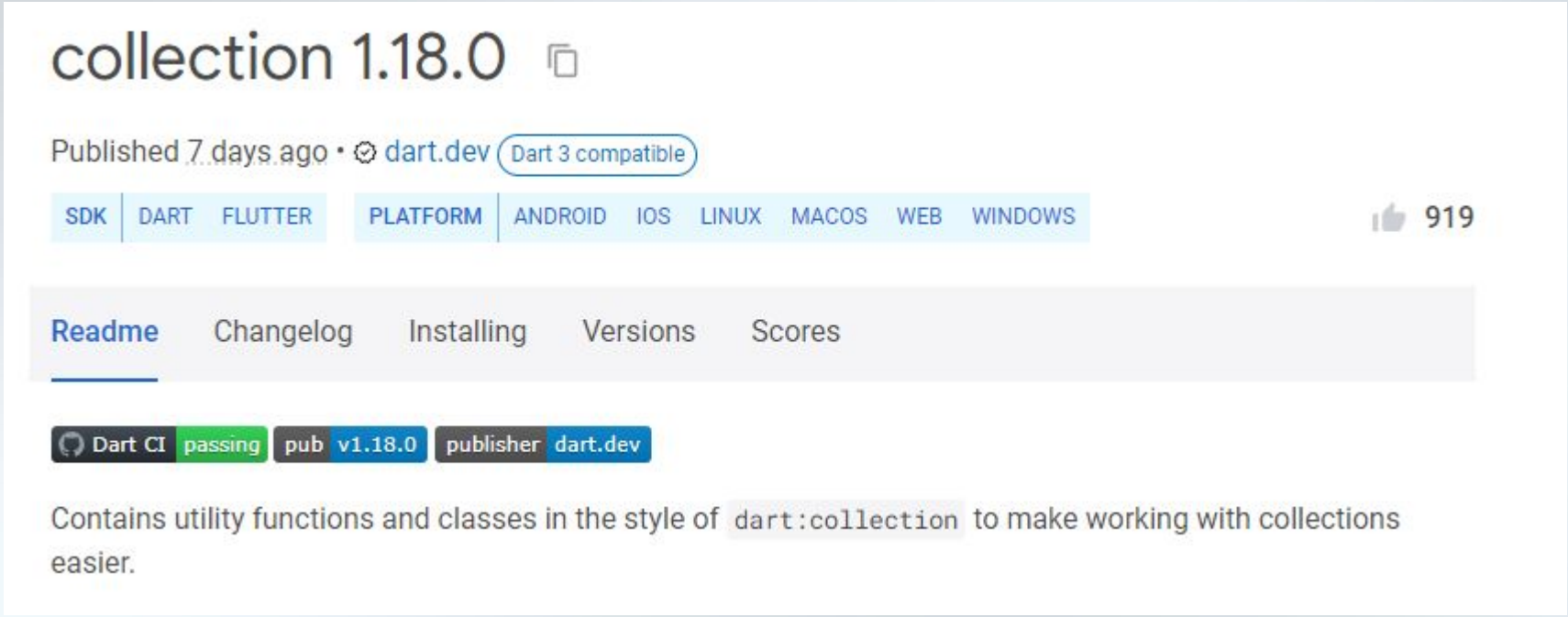
FocusTraversalGroup

```
class CustomTraversalPolicy extends FocusTraversalPolicy {
  @override
  FocusNode? findFirstFocusInDirection(FocusNode currentNode, TraversalDirection direction) {
    // TODO: implement findFirstFocusInDirection
    throw UnimplementedError();
  }

  @override
  bool inDirection(FocusNode currentNode, TraversalDirection direction) {
    // TODO: implement inDirection
    throw UnimplementedError();
  }

  @override
  Iterable<FocusNode> sortDescendants(Iterable<FocusNode> descendants, FocusNode currentNode) {
    // TODO: implement sortDescendants
    throw UnimplementedError();
  }
}
```

```
return Material(
  child: FocusTraversalGroup(
    policy: CustomTraversalPolicy(),
```

FocusTraversalGroup

https://pub.dev/packages/collection



FocusTraversalGroup

```
@override
FocusNode? findFirstFocusInDirection(
    FocusNode currentNode, TraversalDirection direction) {
  if (direction == TraversalDirection.right) {
    final nearest = currentNode.nearestScope!.traversalDescendants;
    final dx = currentNode.offset.dx;
    final focusNodeOnRight =
        nearest.firstWhereOrNull((element) => element.offset.dx > dx);

    if (focusNodeOnRight != null) {
      return focusNodeOnRight;
    } else {
      final bottomFocusNode = nearest.firstWhereOrNull(
          (element) => element.offset.dy > currentNode.offset.dy);

      return bottomFocusNode;
    }
  } else {
    throw UnimplementedError();
  }
}
```

```
@override
bool inDirection(FocusNode currentNode, TraversalDirection direction) {
  final focusNode = findFirstFocusInDirection(currentNode, direction);
  focusNode?.requestFocus();

  return true;
}
```

FocusTraversalGroup

```dart
@override
bool inDirection(FocusNode currentNode, TraversalDirection direction) {
  final focusNode = findFirstFocusInDirection(currentNode, direction);
  focusNode?.requestFocus();

  final scrollable = Scrollable.of(focusNode!.context!);

  scrollable.position.animateTo(
    focusNode.offset.dy,
    duration: const Duration(seconds: 1),
    curve: Curves.ease,
  );

  return true;
}
```

```dart
@override
bool inDirection(FocusNode currentNode, TraversalDirection direction) {
  final focusNode = findFirstFocusInDirection(currentNode, direction);
  focusNode?.requestFocus();

  Scrollable.ensureVisible(focusNode!.context!);

  return true;
}
```

Focus Key Handler

```
 onKey: (node, key) {
   if (key is! RawKeyDownEvent) {
     return KeyEventResult.ignored;
   }

   if (key.hasSubmitIntent) {
     _calculateSubmitActions();
     _callbackHolder?.onSubmit?.call();
     return KeyEventResult.handled;
   }

   return KeyEventResult.ignored;
 },
```

```
void _calculateSubmitActions() {
  _key.currentContext!
      .visitChildElements(_extractSubmitActionFromInteractiveWidget);
}
```

Focus Key Handler

```
void _extractSubmitActionFromInteractiveWidget(Element element) {
  final widget = element.widget;

  if (widget is GestureDetector) {
    _callbackHolder = _CallbackHolder(
        onSubmit: widget.onTap, onLongPress: widget.onLongPress);

    return;
  }

  if (widget is InkWell) {
    _callbackHolder = _CallbackHolder(
        onSubmit: widget.onTap, onLongPress: widget.onLongPress);

    return;
  }

  element.visitChildElements(_extractSubmitActionFromInteractiveWidget);
}
```

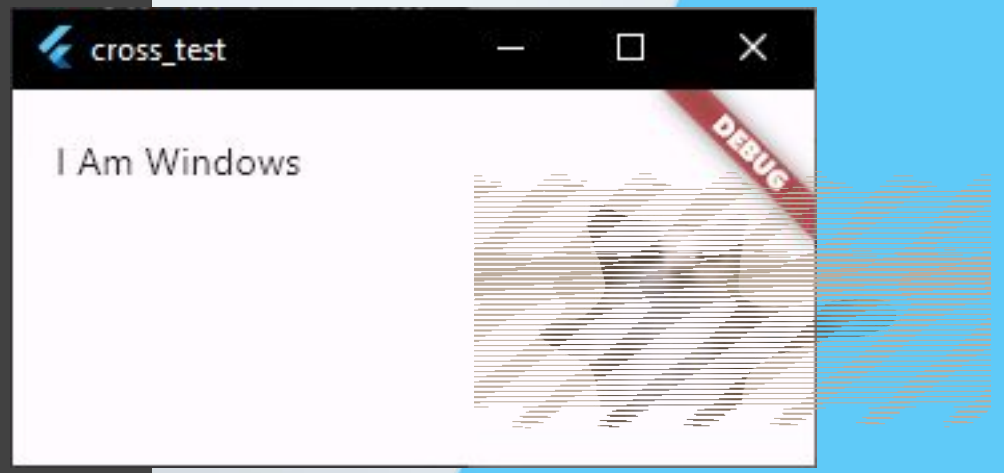
```
extension SubmitAction on RawKeyEvent {
  bool get hasSubmitIntent =>
      logicalKeysTap.contains(logicalKey) || logicalKey.debugName == 'Select';
}
```

Focus Key Handler

```
const logicalKeysTap = [
  LogicalKeyboardKey.select,
  LogicalKeyboardKey.enter,
  LogicalKeyboardKey.space,
  LogicalKeyboardKey.gameButtonA,
];
```

PlatformWidget

```dart
@override
Widget build(BuildContext context) {
  return Material(
    child: FocusTraversalGroup(
      child: Padding(
        padding: const EdgeInsets.all(16),
        child: Builder(
          builder: (context) {
            if (Platform.isAndroid) {
              return const Text('I Am Android');
            }

            if (Platform.isWindows) {
              return const Text('I Am Windows');
            }

            if (kIsWeb) {
              return const Text('I Am Web');
            }
          },
        ),
      ),
    ),
  );
}
```



PlatformWidget

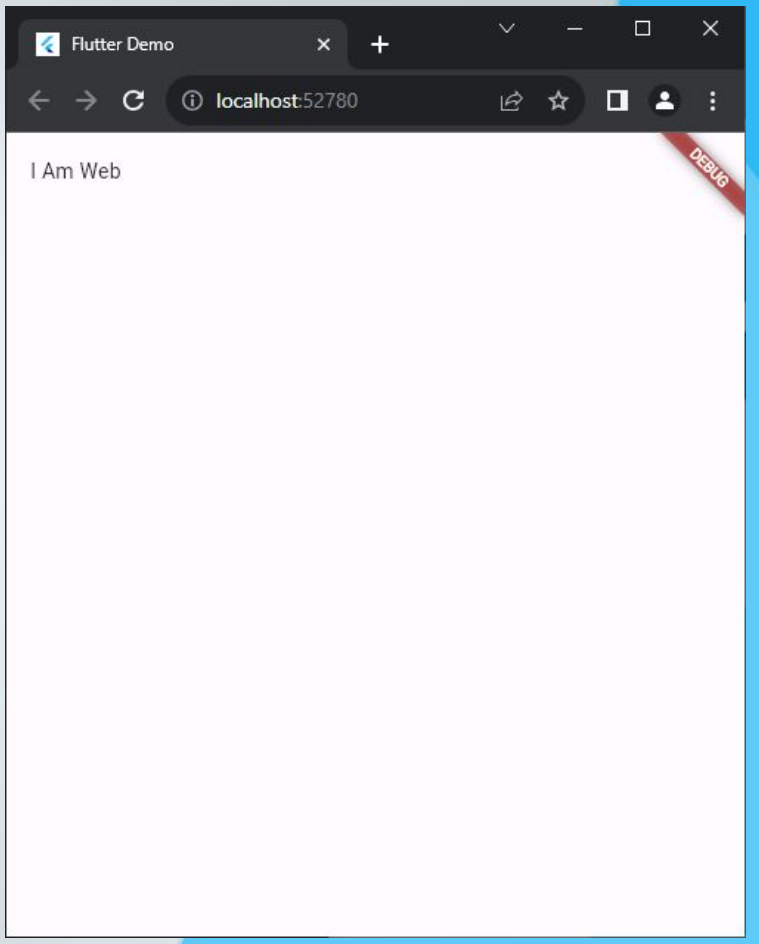
```dart
import 'dart:io';
```

PlatformWidget

```dart
import 'dart:io';

class ExtendedPlatform {
  static bool get isAndroid => Platform.isAndroid;
  static bool get isIOS => Platform.isIOS;
  static bool get isWindows => Platform.isWindows;
  static bool get isWeb => false;
}
```

```dart
abstract class ExtendedPlatform {
  static bool get isAndroid => false;
  static bool get isIOS => false;
  static bool get isWindows => false;
  static bool get isWeb => true;
}
```

```dart
export 'package:cross_test/extended_platform/extended_platform_other.dart'
    if (dart.library.html)
'package:cross_test/extended_platform/extended_platform_web.dart';
```



PlatformWidget

```dart
class PlatformBuilder<T> extends StatelessWidget {
  final T? android;
  final T? iOS;
  final T? web;
  final T? windows;
  final T? other;
  final Widget Function(BuildContext, Widget?, T) builder;

  final Widget? child;

  const PlatformBuilder({
    required this.builder,
    this.android,
    this.iOS,
    this.web,
    this.windows,
    this.other,
    this.child,
    super.key,
  });

  @override
  Widget build(BuildContext context) {
    if (ExtendedPlatform.isAndroid && android != null) {
      return builder.call(context, child, android!);
    }

    //... other
    return builder.call(context, child, other!);
  }
}
```

```dart
@override
Widget build(BuildContext context) {
  return Material(
    child: FocusTraversalGroup(
      child: Padding(
        padding: const EdgeInsets.all(16),
        child: PlatformBuilder(
          android: 'I am Android',
          other: 'Other platform',
          builder: (context, child, data) {
            return Text(data);
          },
        ),
      ),
    ),
  );
}
```

PlatformWidget

```dart
import 'dart:io';

import 'package:device_info_plus/device_info_plus.dart';

class ExtendedPlatform {
  static Future<void> initialize() async {
    if (Platform.isAndroid) {
      DeviceInfoPlugin deviceInfo = DeviceInfoPlugin();
      AndroidDeviceInfo androidInfo = await deviceInfo.androidInfo;
      isTv =
          androidInfo.systemFeatures.contains('android.software.leanback_only');
    } else if (Platform.isIOS || Platform.isWindows) {
      isTv = false;
    } else {
      isTv = Platform.isLinux;
    }
  }

  static bool isTv = false;

  static bool isTizen =
      Platform.isLinux && !Platform.isAndroid && !Platform.isIOS;
}
```

Работа с мышью

MouseRegion – это виджет, который позволяет отслеживать, наведена мышь на какой-либо элемент, или нет. Работает на всех системах, где есть мышь

Listener – это виджет, который позволяет отслеживать любого рода взаимодействия с виджетом, используя курсор

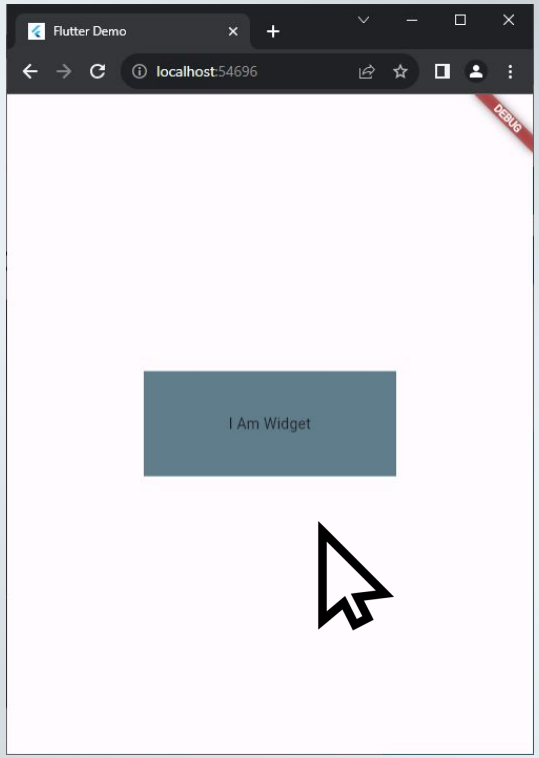
GestureDetector – это виджет, который позволяет отслеживать взаимодействие с виджетом как мышкой, так и касаниями

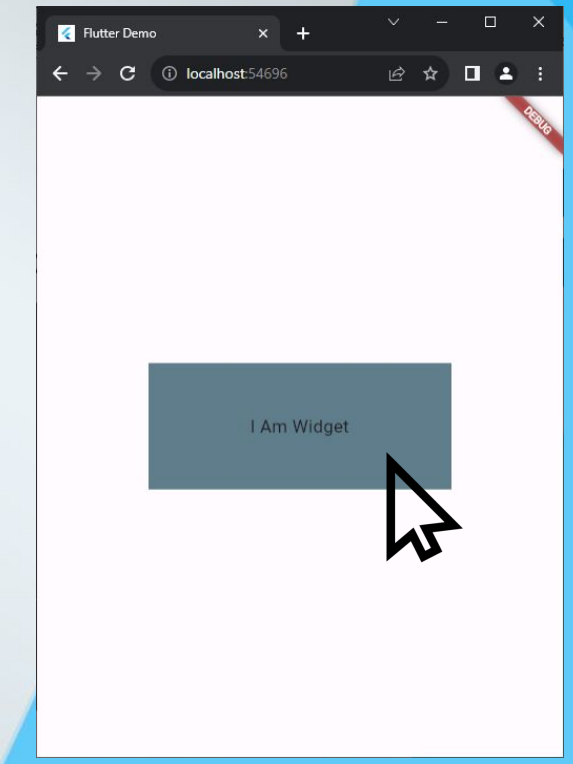
Работа с мышью

```
class HoverableWidget extends StatefulWidget {
  final Widget Function(BuildContext, Widget?, bool) builder;
  final Widget? child;

  const HoverableWidget({
    required this.builder,
    this.child,
    super.key,
  });

  @override
  State<HoverableWidget> createState() => _HoverableWidgetState();
}

class _HoverableWidgetState extends State<HoverableWidget> {
  bool _isHovered = false;
  @override
  Widget build(BuildContext context) {
    return MouseRegion(
      onEnter: (event) {
        setState(() {
          _isHovered = true;
        });
      },
      onExit: (event) {
        setState(() {
          _isHovered = false;
        });
      },
      child: widget.builder(context, widget.child, _isHovered),
    );
  }
}
```

Работа с мышью

```dart
@override
Widget build(BuildContext context) {
  return Material(
    child: Center(
      child: HoverableWidget(
        builder: (context, child, isHovered) {
          return AnimatedScale(
            scale: isHovered ? 1.2 : 1.0,
            duration: const Duration(milliseconds: 200),
            child: child,
          );
        },
        child: Container(
          width: 240,
          height: 100,
          color: Colors.blueGrey,
          child: const Center(child: Text('I Am Widget')),
        ),
      ),
    ),
  );
}
```



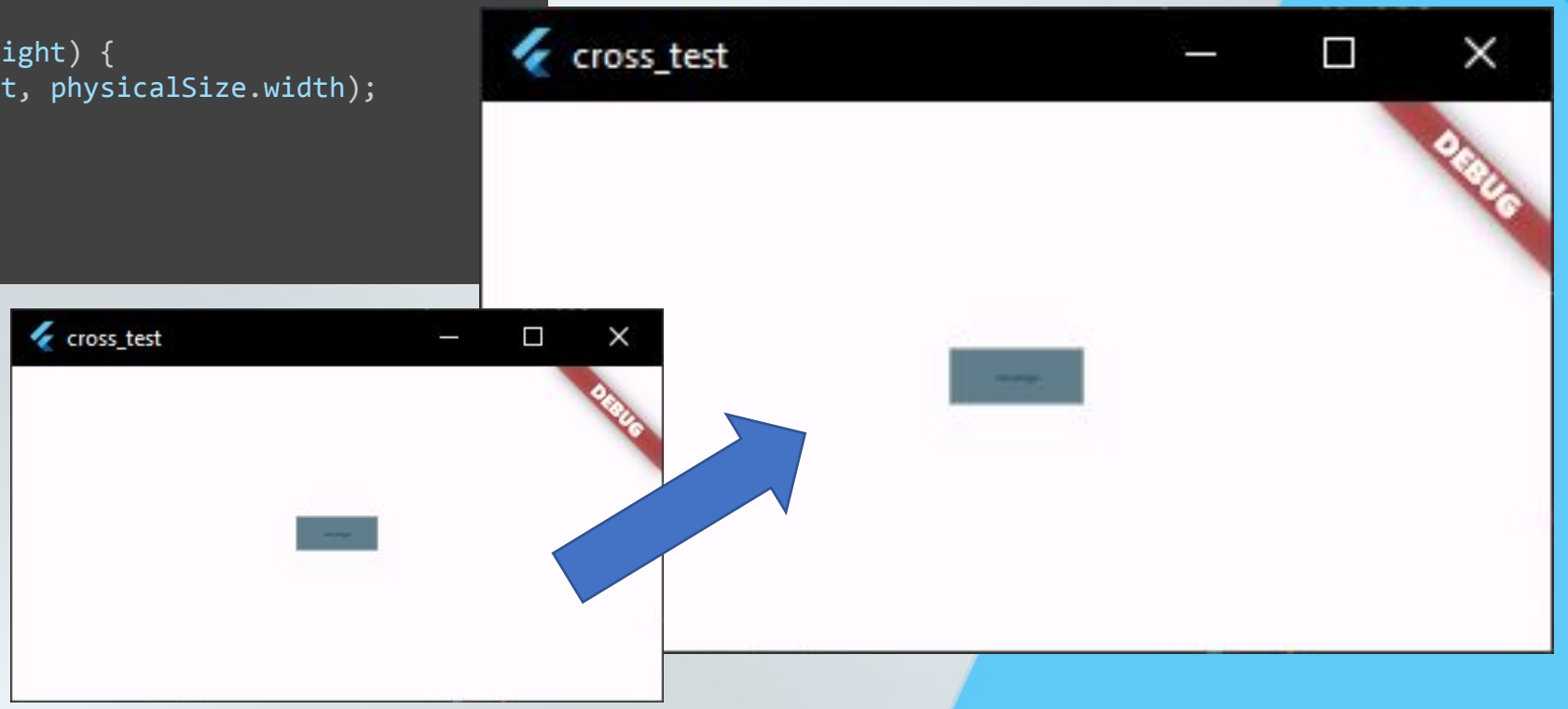


Масштабирование приложения

```
class ScaleWidget extends StatelessWidget {
  final Widget child;

  const ScaleWidget({
    required this.child,
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final ratio = _getScaleRatio(context);

    return FractionallySizedBox(
      widthFactor: 1 / ratio,
      heightFactor: 1 / ratio,
      child: Transform.scale(
        scale: ratio,
        child: Center(
          child: SizedBox(
            width: 1920,
            height: 1920 *
                MediaQuery.of(context).size.height /
                MediaQuery.of(context).size.width,
            child: child,
          ),
        ),
      ),
    );
  }
 }
```

Масштабирование приложения

```dart
double _getScaleRatio(BuildContext context) {
  return _getWindowSizeInLogicalPixels(context).width / 1920;
}

Size _getWindowSizeInLogicalPixels(BuildContext context) {
  var physicalSize = View.of(context).physicalSize;
  var pixelRatio = View.of(context).devicePixelRatio;

  if (physicalSize.width < physicalSize.height) {
    physicalSize = Size(physicalSize.height, physicalSize.width);
  }

  return physicalSize / pixelRatio;
}
```



Урок 12. Кросс-платформенные решения

- Научились создавать виджеты для адаптивного дизайна
- Научились работать с виджетом Focus
- Научились реализовывать собственные политики фокусировки виджетов
- Научились обрабатывать нажатия кнопок
- Научились разделять дизайн по платформам
- Научились виджеты взаимодействовать с курсором мыши
- Научились работать с масштабированием на ТВ

Q & A