**LÝ THUYẾT BUỔI 7**

Họ và tên: Lê Thị Bảo Yến

MSSV: 0850080057

Lớp: 08\_ĐH\_CNPM

# **AnimatedWidget**

AnimatedWidget là một lớp trong Flutter được sử dụng để tạo các widget có hiệu ứng chuyển động. Đây là một lớp cơ bản cho các widget có khả năng thay đổi thuộc tính của chúng một cách hiệu quả. Lớp này thường được sử dụng kết hợp với widget AnimatedBuilder.

Lớp AnimatedWidget là một stateful widget, nhận đầu vào là một đối tượng Animation và rebuild lại cây widget mỗi khi giá trị của animation thay đổi. Đối tượng Animation có thể được tạo bằng bất kỳ lớp animation nào được tích hợp sẵn trong Flutter, chẳng hạn như Tween, CurvedAnimation, hoặc AnimationController.

Để sử dụng AnimatedWidget, bạn cần tạo một lớp con của nó và triển khai phương thức build(). Phương thức này được gọi mỗi khi giá trị của animation thay đổi và nó nên trả về cây widget mà bạn muốn động.

Dưới đây là ví dụ về việc sử dụng AnimatedWidget để tạo hiệu ứng mờ dần cho một widget:

class FadeTransitionWidget extends AnimatedWidget {

FadeTransitionWidget({Key key, Animation<double> animation})

: super(key: key, listenable: animation);

@override

Widget build(BuildContext context) {

final Animation<double> animation = listenable;

return Opacity(

opacity: animation.value,

child: Text('Hello, world!'),

);

}

}

Trong ví dụ này, FadeTransitionWidget là một lớp con của AnimatedWidget nhận đầu vào là một đối tượng Animation<double>. Phương thức build()trả về một widget Opacity sử dụng giá trị animation để điều khiển độ mờ của widget con (trong trường hợp này, một widget Text). Mỗi khi giá trị animation thay đổi, phương thức build() được gọi và cây widget được rebuild với giá trị độ mờ mới.

# **AnimatedBuilder**

AnimatedBuilder là một widget trong Flutter được sử dụng để xây dựng các widget có hiệu ứng chuyển động. Widget này cho phép bạn tái sử dụng đoạn mã xử lý animation và giúp tối ưu hóa hiệu suất vì chỉ rebuild lại widget con.

AnimatedBuilder có thể được sử dụng để xây dựng các widget có khả năng thay đổi thuộc tính của chúng, chẳng hạn như độ mờ, vị trí, kích thước, hoặc màu sắc. Widget này cho phép bạn tái sử dụng đoạn mã xử lý animation, giúp tối ưu hóa hiệu suất vì chỉ rebuild lại widget con.

Để sử dụng AnimatedBuilder, bạn cần tạo một đối tượng Animation và một hàm xử lý gọi lại (callback) để xây dựng widget. Hàm xử lý gọi lại sẽ nhận đầu vào là giá trị animation và trả về widget cần xây dựng. AnimatedBuilder sẽ rebuild lại widget mỗi khi giá trị animation thay đổi.

Ví dụ sau đây sử dụng AnimatedBuilder để tạo hiệu ứng mờ dần cho một widget:

class FadeTransitionWidget extends StatefulWidget {

@override

\_FadeTransitionWidgetState createState() => \_FadeTransitionWidgetState();

}

class \_FadeTransitionWidgetState extends State<FadeTransitionWidget> with SingleTickerProviderStateMixin {

AnimationController \_controller;

Animation<double> \_animation;

@override

void initState() {

super.initState();

\_controller = AnimationController(

vsync: this,

duration: Duration(seconds: 2),

);

\_animation = Tween<double>(

begin: 0.0,

end: 1.0,

).animate(\_controller);

\_controller.forward();

}

@override

Widget build(BuildContext context) {

return AnimatedBuilder(

animation: \_animation,

builder: (BuildContext context, Widget child) {

return Opacity(

opacity: \_animation.value,

child: Text('Hello, world!'),

);

},

);

}

@override

void dispose() {

\_controller.dispose();

super.dispose();

}

}

Trong ví dụ này, AnimatedBuilder được sử dụng để xây dựng widget có hiệu ứng mờ dần. FadeTransitionWidget là một stateful widget và nó tạo ra một đối tượng AnimationController để điều khiển animation. Đối tượng Animation được tạo bằng `Tween`, và giá trị animation được truyền vào AnimatedBuilder để xây dựng widget. Mỗi khi giá trị animation thay đổi, AnimatedBuilder sẽ rebuild lại widget con.

# **AnimationController**

AnimationController là một lớp trong Flutter được sử dụng để điều khiển các đối tượng animation. Nó cho phép bạn tạo ra các giá trị animation và điều khiển tốc độ chuyển động của chúng.

Để tạo một đối tượng AnimationController, bạn cần chỉ định một đối tượng vsync để đồng bộ hóa animation với các tín hiệu về khung hình (frame) được cung cấp bởi Ticker. vsync là một đối tượng TickerProvider, thường là State của một StatefulWidget.

Bạn cũng nên chỉ định thời gian chuyển động của animation bằng cách sử dụng thuộc tính duration. Sau khi tạo đối tượng AnimationController, bạn có thể bắt đầu animation bằng cách gọi phương thức forward() hoặc reverse(), tùy thuộc vào hướng chuyển động của animation.

Dưới đây là một ví dụ về việc sử dụng `AnimationController` để tạo animation di chuyển của một widget:

Ví dụ:

class MoveTransitionWidget extends StatefulWidget {

@override

\_MoveTransitionWidgetState createState() => \_MoveTransitionWidgetState();

}

class\_MoveTransitionWidgetState extends State<MoveTransitionWidget> with SingleTickerProviderStateMixin {

AnimationController \_controller;

Animation<Offset> \_animation;

@override

void initState() {

super.initState();

\_controller = AnimationController(

vsync: this,

duration: Duration(seconds: 2),

);

\_animation = Tween<Offset>(

begin: Offset(0.0, 0.0),

end: Offset(1.0, 1.0),

).animate(\_controller);

\_controller.forward();

}

@override

Widget build(BuildContext context) {

return SlideTransition(

position: \_animation,

child: Text('Hello, world!'),

);

}

@override

void dispose() {

\_controller.dispose();

super.dispose();

}

}

Trong ví dụ này, AnimationController được sử dụng để tạo animation di chuyển của một widget. MoveTransitionWidget là một stateful widget và nó tạo ra một đối tượng AnimationController để điều khiển animation. Đối tượng Animation được tạo bằng Tween, và giá trị animation được truyền vào SlideTransition để di chuyển widget. Mỗi khi giá trị animation thay đổi, widget sẽ được rebuild lại với vị trí mới.

# **CuvedAnimation**

CurvedAnimation là một lớp trong Flutter được sử dụng để tạo ra các giá trị animation với một hàm curve (đường cong) xác định. Hàm curve này giúp bạn điều chỉnh tốc độ chuyển động của animation trong suốt quá trình chạy.

Hàm curve của CurvedAnimation là một đối tượng Curve, và bạn có thể sử dụng các đối tượng curve tích hợp sẵn như Curves.linear, Curves.easeIn, Curves.easeOut, Curves.easeInOut, hoặc tạo một curve tùy chỉnh.

Để sử dụng CurvedAnimation, bạn cần tạo một đối tượng Animation (thường là AnimationController) để điều khiển animation, và sử dụng CurvedAnimation để tạo ra các giá trị animation với hàm curve được chỉ định. Sau đó, bạn có thể sử dụng các giá trị animation này để điều khiển thuộc tính của widget.

Dưới đây là một ví dụ về việc sử dụng CurvedAnimation để tạo animation di chuyển của một widget với hàm curve tùy chỉnh:

class MoveTransitionWidget extends StatefulWidget {

@override

\_MoveTransitionWidgetState createState() => \_MoveTransitionWidgetState();

}

class \_MoveTransitionWidgetState extends State<MoveTransitionWidget> with SingleTickerProviderStateMixin {

AnimationController \_controller;

Animation<Offset> \_animation;

@override

void initState() {

super.initState();

\_controller = AnimationController(

vsync: this,

duration: Duration(seconds: 2),

);

\_animation = CurvedAnimation(

parent: \_controller,

curve: Curves.elasticOut,

).drive(

Tween<Offset>(

begin: Offset(0.0, 0.0),

end: Offset(1.0, 1.0),

),

);

\_controller.forward();

}

@override

Widget build(BuildContext context) {

return SlideTransition(

position: \_animation,

child: Text('Hello, world!'),

);

}

@override

void dispose() {

\_controller.dispose();

super.dispose();

}

}

Trong ví dụ này, CurvedAnimation được sử dụng để tạo animation di chuyển của một widget với hàm curve tùy chỉnh Curves.elasticOut. MoveTransitionWidget là một stateful widget và nó tạo ra một đối tượng AnimationController để điều khiển animation. Đối tượng CurvedAnimation được tạo bằng Tween, và giá trị animation được truyền vào SlideTransition để di chuyển widget. Mỗi khi giá trị animation thay đổi, widget sẽ được rebuild lại với vị trí mới được điều chỉnh bởi hàm curve tùy chỉnh.

# **Hero**

Hero là một widget trong Flutter được sử dụng để tạo hiệu ứng chuyển động mượt mà giữa các màn hình. Widget này cho phép bạn định nghĩa một `Hero` trên một màn hình và một Hero tương ứng trên màn hình khác, và nó sẽ tự động tạo hiệu ứng chuyển động khi chuyển từ màn hình này sang màn hình khác.

Để sử dụng Hero, bạn cần định nghĩa một Hero trên một màn hình bằng cách đặt widget này vào trong một Hero widget và cung cấp một tag định danh cho Hero. Sau đó, bạn cần định nghĩa một `Hero` tương ứng trên màn hình khác bằng cách đặt widget này vào trong một `Hero` widget và cung cấp cùng một `tag` định danh.

Khi chuyển đổi từ màn hình này sang màn hình khác, `Hero` sẽ tự động tạo hiệu ứng chuyển động mượt mà giữa hai widget với cùng `tag` định danh. Hiệu ứng chuyển động này có thể bao gồm các phép biến đổi kích thước, vị trí, độ mờ, hoặc màu sắc của các widget.

Dưới đây là một ví dụ về việc sử dụng `Hero` để tạo hiệu ứng chuyển động mượt mà giữa hai màn hình:

**Màn hình 1:**

class Screen1 extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

body: Center(

child: Hero(

tag: 'heroTag',

child: Image.network('https://picsum.photos/250?image=9'),

),

),

floatingActionButton: FloatingActionButton(

onPressed: () {

Navigator.push(

context,

MaterialPageRoute(builder: (context) => Screen2()),

);

},

child: Icon(Icons.arrow\_forward),

),

);

}

}

**Màn hình 2:**

class Screen2 extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

body: Center(

child: Hero(

tag: 'heroTag',

child: Image.network('https://picsum.photos/250?image=10'),

),

),

floatingActionButton: FloatingActionButton(

onPressed: () {

Navigator.pop(context);

},

child: Icon(Icons.arrow\_back),

),

);

}

}

Trong ví dụ này, Hero được sử dụng để tạo hiệu ứng chuyển động mượt mà giữa hai màn hình. Trên màn hình 1, chúng ta định nghĩa một Hero với tag định danh là heroTag và chứa một Image widget. Trên màn hình 2, chúng ta cũng định nghĩa một Hero với cùng tag định danh và chứa một Image widget khác. Khi chuyển từ màn hình 1 sang màn hình 2, Hero sẽ tự động tạo hiệu ứng chuyển động mượt mà giữa hai Image widget với cùng `tag` định danh.

# **Opacity**

Opacity là một widget trong Flutter được sử dụng để thay đổi độ mờ của một widget. Widget này cho phép bạn điều chỉnh độ mờ của một widget từ 0.0 (hoàn toàn trong suốt) đến 1.0 (hoàn toàn không trong suốt).

Để sử dụng Opacity, bạn cần đặt widget cần thay đổi độ mờ vào trong một Opacity widget và cung cấp giá trị opacity. Giá trị này là một số thực nằm trong khoảng từ 0.0 đến 1.0, và nó xác định độ mờ của widget.

Dưới đây là một ví dụ về việc sử dụng Opacity để thay đổi độ mờ của một widget:

class OpacityWidget extends StatefulWidget {

@override

\_OpacityWidgetState createState() => \_OpacityWidgetState();

}

class \_OpacityWidgetState extends State<OpacityWidget> {

double \_opacity = 1.0;

void \_toggleOpacity() {

setState(() {

\_opacity = \_opacity == 1.0 ? 0.0 : 1.0;

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Opacity Widget')),

body: Center(

child: Opacity(

opacity: \_opacity,

child: Image.network('https://picsum.photos/250?image=9'),

),

),

floatingActionButton: FloatingActionButton(

onPressed: \_toggleOpacity,

child: Icon(Icons.opacity),

),

);

}

}

Trong ví dụ này, Opacity được sử dụng để thay đổi độ mờ của một Image widget. Khi giá trị \_opacity là 1.0, Image widget sẽ không bị mờ. Khi giá trị \_opacity là 0.0, Image widget sẽ hoàn toàn trong suốt. Khi người dùng nhấn vào floating action button, giá trị \_opacity sẽ được đảo ngược, dẫn đến thay đổi độ mờ của Image widget.