**1. AnimatedWidget**

Animation trong Flutter được tạo ra bằng cách (Flutter) vẽ lại liên tục các widget (khoảng 60 fps) làm cho chúng ta thấy các widget này thay đổi một cách mượt mà.

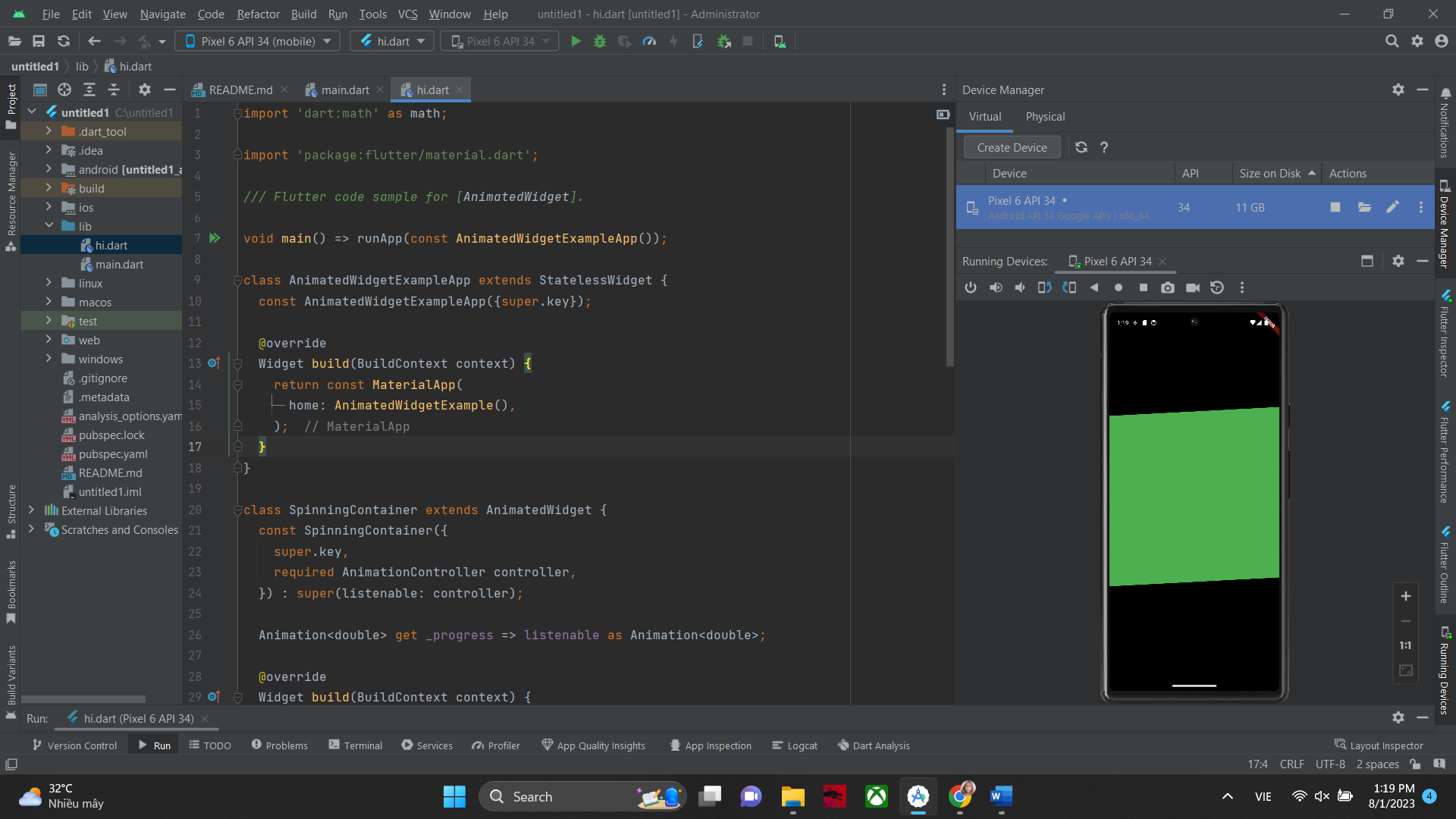
Flutter cung cấp cho chúng ta một số class để implement animation

* Animation<T>: abstract class, định nghĩa sự thay đổi thuộc tính của widget.
* AnimationController: sinh ra 1 giá trị mỗi new frame (mặc định là từ 0.0 đến 1.0 theo hàm tuyến tính) trong 1 khoảng thời gian.

AnimatedWidget là abstract class, để sử dụng ta cần extends nó và truyền vào đối tượng Listenable (AnimationController) và override lại hàm build(). Mỗi khi AnimationController sinh ra một giá trị mới thì AnimatedWiget sẽ tự build lại. AnimatedBuilder là subclass của AnimatedWidget.

AnimatedWidget được sử dụng phổ biến nhất với các đối tượng Animation , là Listenable , nhưng nó cũng có thể được sử dụng với bất kỳ Listenable nào , bao gồm cả ChangeNotifier và ValueNotifier .

AnimatedWidget hữu ích nhất cho các widget không trạng thái. Để sử dụng AnimatedWidget , hãy phân lớp nó và triển khai chức năng xây dựng.



**Mối quan hệ với ImplicitlyAnimatedWidget s**

AnimatedWidget s (và các lớp con của chúng) lấy một đối số Listenable rõ ràng làm đối số, thường là Animation bắt nguồn từ AnimationController.

Trong hầu hết các trường hợp, vòng đời của AnimationController đó phải được quản lý thủ công bởi nhà phát triển. Ngược lại, ImplicitlyAnimatedWidget s (và các lớp con của chúng) tự động quản lý AnimationController bên trong của riêng chúng, làm cho các lớp đó dễ sử dụng hơn vì nhà phát triển không phải cung cấp Animation bên ngoài.

Nếu bạn chỉ cần đặt giá trị đích cho hoạt ảnh và định cấu hình thời lượng/đường cong của hoạt ảnh, hãy cân nhắc sử dụng (một lớp con của) ImplicitlyAnimatedWidgets thay vì (một lớp con của) lớp này.

**Các vật dụng hoạt hình phổ biến**

Một số tiện ích động đi kèm với khung. Chúng thường được đặt tên FooTransition, đâu Foo là tên của phiên bản không hoạt hình của tiện ích con đó. Không nên nhầm lẫn các lớp con của lớp này với các lớp con của ImplicitlyAnimatedWidget (xem ở trên), thường được đặt tên là AnimatedFoo. Các vật dụng hoạt hình thường được sử dụng bao gồm:

* ListenableBuilder , sử dụng mẫu trình tạo hữu ích cho các trường hợp sử dụng Listenable phức tạp .
* AnimatedBuilder , sử dụng mẫu trình tạo hữu ích cho các trường hợp sử dụng Hoạt hình phức tạp .
* AlignTransition , là phiên bản động của Align .
* DecoredBoxTransition , là phiên bản hoạt hình của DecoredBox .
* DefaultTextStyleTransition , là phiên bản động của DefaultTextStyle .
* PositionedTransition , là phiên bản hoạt hình của Positioned .
* RelativePositionedTransition , là phiên bản hoạt hình của Positioned .
* RotationTransition , tạo hiệu ứng xoay của tiện ích con.
* ScaleTransition , tạo hoạt ảnh theo tỷ lệ của tiện ích con.
* SizeTransition , tạo hoạt ảnh cho kích thước của chính nó.
* SlideTransition , làm sinh động vị trí của một tiện ích con so với vị trí bình thường của nó.
* FadeTransition , là phiên bản động của Opacity .
* AnimatedModalBarrier , là phiên bản hoạt hình của ModalBarrier .

**2. AnimatedBuilder**

Một tiện ích có mục đích chung để tạo hoạt ảnh. AnimatedBuilder hữu ích cho các vật dụng phức tạp hơn muốn bao gồm hoạt ảnh như một phần của chức năng xây dựng lớn hơn. Để sử dụng AnimatedBuilder , hãy xây dựng tiện ích con và chuyển cho nó một chức năng trình tạo.

Đối với các trường hợp đơn giản không có trạng thái bổ sung, hãy cân nhắc sử dụng AnimatedWidget.

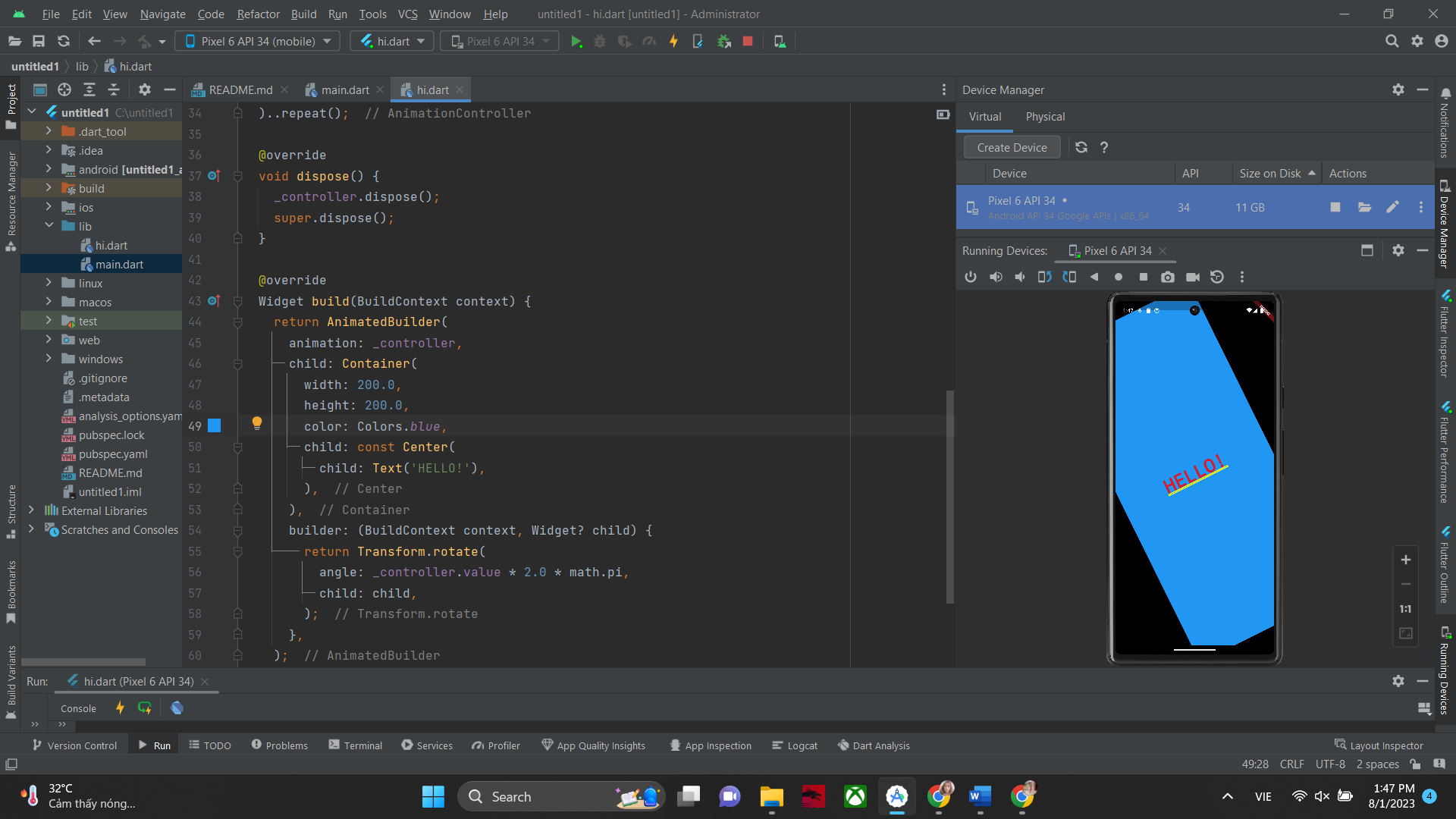
Bất chấp tên gọi, AnimatedBuilder không giới hạn ở Animation s, bất kỳ loại phụ nào của Listenable (chẳng hạn như ChangeNotifier hoặc ValueNotifier ) đều có thể được sử dụng để kích hoạt quá trình xây dựng lại.

**Tối ưu hóa hiệu suất**

Nếu hàm xây dựng chứa một cây con không phụ thuộc vào hoạt ảnh được chuyển đến hàm tạo, thì việc xây dựng cây con đó một lần sẽ hiệu quả hơn thay vì xây dựng lại nó trên mỗi dấu tích hoạt ảnh.

Nếu một cây con dựng sẵn được truyền dưới dạng tham số con , thì AnimatedBuilder sẽ chuyển nó trở lại chức năng của trình tạo để nó có thể được tích hợp vào bản dựng.

Việc sử dụng công cụ con dựng sẵn này là hoàn toàn tùy chọn nhưng có thể cải thiện hiệu suất đáng kể trong một số trường hợp và do đó là một phương pháp hay.



**3. AnimationController**

Một bộ điều khiển cho một hình ảnh động.

Lớp này cho phép bạn thực hiện các tác vụ như:

Phát hoạt ảnh tiến hoặc lùi hoặc dừng hoạt ảnh.

Đặt hoạt ảnh thành một giá trị cụ thể .

Xác định các giá trị UpperBound và LowerBound của hoạt ảnh.

Tạo hiệu ứng hoạt hình bằng cách sử dụng mô phỏng vật lý.

Theo mặc định, AnimationController tạo tuyến tính các giá trị nằm trong khoảng từ 0,0 đến 1,0 trong một khoảng thời gian nhất định. Bộ điều khiển hoạt hình tạo ra một giá trị mới bất cứ khi nào thiết bị chạy ứng dụng của bạn sẵn sàng hiển thị một khung hình mới

**Nhà cung cấp mã chứng khoán**

Một AnimationController cần một TickerProvider , được cấu hình bằng cách sử dụng vsync đối số trên hàm tạo.

Giao diện TickerProvider mô tả một nhà máy cho các đối tượng Ticker . Ticker là một đối tượng biết cách đăng ký chính nó với SchedulerBinding và kích hoạt một cuộc gọi lại mỗi khung. Lớp AnimationController sử dụng Ticker để chuyển qua hoạt ảnh mà nó điều khiển.

Nếu một AnimationController đang được tạo từ một State , thì State có thể sử dụng các lớp TickerProviderStateMixin và SingleTickerProviderStateMixin để triển khai giao diện TickerProvider . Lớp TickerProviderStateMixin luôn hoạt động cho mục đích này; SingleTickerProviderStateMixin hiệu quả hơn một chút trong trường hợp lớp chỉ cần một Ticker

Khung kiểm tra tiện ích Đối tượng WidgetTester có thể được sử dụng làm nhà cung cấp mã đánh dấu trong bối cảnh kiểm tra. Trong các ngữ cảnh khác, bạn sẽ phải chuyển một TickerProvider từ cấp cao hơn (ví dụ: gián tiếp từ một Trạng thái trộn lẫn trong TickerProviderStateMixin ) hoặc tạo một lớp con TickerProvider tùy chỉnh .

Vòng đời

Một AnimationController nên được loại bỏ d khi nó không còn cần thiết nữa. Điều này làm giảm khả năng rò rỉ. Khi được sử dụng với StatefulWidget, AnimationController thường được tạo trong phương thức State.initState và sau đó được xử lý trong phương thức State.dispose .

Sử dụng Future s với AnimationController

Các phương thức bắt đầu hoạt ảnh trả về một đối tượng TickerFuture hoàn thành khi hoạt ảnh hoàn thành thành công và không bao giờ đưa ra lỗi; nếu hoạt ảnh bị hủy, tương lai sẽ không bao giờ hoàn thành. Đối tượng này cũng có thuộc tính TickerFuture.orCancel trả về một tương lai hoàn thành khi hoạt ảnh hoàn tất thành công và hoàn thành với lỗi khi hoạt ảnh bị hủy bỏ.

Điều này có thể được sử dụng để viết mã như fadeOutAndUpdateStatephương pháp dưới đây.

**4. Lớp CurvedAnimation**

Hoạt ảnh áp dụng đường cong cho hoạt ảnh khác.

CurvedAnimation hữu ích khi bạn muốn áp dụng Đường cong phi tuyến tính cho đối tượng hoạt ảnh, đặc biệt nếu bạn muốn các đường cong khác nhau khi hoạt ảnh tiến lên so với khi hoạt ảnh lùi lại.

Tùy thuộc vào đường cong đã cho, đầu ra của CurvedAnimation có thể có phạm vi rộng hơn đầu vào của nó. Ví dụ: các đường cong đàn hồi như Curves.elasticIn sẽ vượt quá hoặc thiếu đáng kể phạm vi mặc định từ 0,0 đến 1,0.

**5. Hero**

**Hero amination tiêu chuẩn**

Hoạt hình anh hùng tiêu chuẩn đưa anh hùng từ tuyến đường này sang tuyến đường mới, thường hạ cánh ở một vị trí khác và với kích thước khác.

**Hero amination xuyên tâm**

Trong hoạt hình anh hùng hướng tâm , khi anh hùng bay giữa các tuyến đường, hình dạng của anh hùng dường như thay đổi từ hình tròn sang hình chữ nhật.

**Cấu trúc cơ bản của Hero amination**

Được triển khai bằng hai Hero tiện ích: một mô tả tiện ích trong tuyến nguồn và một mô tả tiện ích khác trong tuyến đích. Từ quan điểm của người dùng, anh hùng dường như được chia sẻ và chỉ lập trình viên mới cần hiểu chi tiết triển khai này. Mã Hero amination có cấu trúc như sau:

Xác định tiện ích Anh hùng bắt đầu, được gọi là anh hùng nguồn . Anh hùng chỉ định biểu diễn đồ họa của nó (thường là hình ảnh) và thẻ nhận dạng và nằm trong cây tiện ích hiện được hiển thị như được xác định bởi tuyến nguồn.

Xác định tiện ích Anh hùng kết thúc, được gọi là anh hùng đích . Anh hùng này cũng chỉ định biểu diễn đồ họa của nó và cùng một thẻ với anh hùng nguồn. Điều cần thiết là cả hai tiện ích anh hùng đều được tạo bằng cùng một thẻ , thường là một đối tượng đại diện cho dữ liệu cơ bản. Để có kết quả tốt nhất, các anh hùng nên có các cây phụ tùng gần như giống hệt nhau.

Tạo một tuyến đường chứa anh hùng đích. Tuyến đích xác định cây widget tồn tại ở cuối hoạt ảnh.

Kích hoạt hoạt ảnh bằng cách đẩy tuyến đích trên ngăn xếp của Bộ điều hướng. Thao tác đẩy và bật Bộ điều hướng kích hoạt hoạt ảnh anh hùng cho từng cặp anh hùng có thẻ phù hợp trong tuyến đường nguồn và đích.

Flutter tính toán tween tạo hoạt ảnh cho các giới hạn của Anh hùng từ điểm bắt đầu đến điểm cuối (nội suy kích thước và vị trí) và thực hiện hoạt ảnh trong một lớp phủ.

**6. Opacity**

Một tiện ích làm cho phần tử con của nó trong suốt một phần.

Lớp này vẽ phần tử con của nó vào một bộ đệm trung gian và sau đó trộn phần tử con trở lại với cảnh trong suốt một phần.

Đối với các giá trị của độ mờ khác 0,0 và 1,0, lớp này tương đối đắt tiền vì nó yêu cầu vẽ phần tử con vào một bộ đệm trung gian. Đối với giá trị 0,0, đứa trẻ hoàn toàn không được vẽ. Đối với giá trị 1.0, phần tử con được vẽ ngay lập tức mà không cần bộ đệm trung gian.

Sự hiện diện của bộ đệm trung gian có nền trong suốt theo mặc định có thể khiến một số tiện ích con hoạt động khác đi. Ví dụ: một công cụ con BackdropFilter sẽ chỉ có thể áp dụng bộ lọc của nó cho nội dung giữa tiện ích con này và công cụ con phông nền và có thể yêu cầu điều chỉnh thuộc tính BackdropFilter.blendMode để tạo ra kết quả mong muốn.

**Cân nhắc hiệu suất cho hoạt hình độ mờ**

Tạo hoạt ảnh cho tiện ích Opacity trực tiếp khiến tiện ích (và có thể là cây con của nó) xây dựng lại từng khung, điều này không hiệu quả lắm. Thay vào đó, hãy xem xét sử dụng một trong những vật dụng thay thế này:

AnimatedOpacity , sử dụng hoạt ảnh bên trong để tạo hiệu ứng mờ đục hiệu quả.

FadeTransition , sử dụng hoạt ảnh được cung cấp để tạo hiệu ứng mờ đục hiệu quả.

Hình ảnh trong suốt

Nếu chỉ cần kết hợp một Hình ảnh hoặc Màu duy nhất với độ mờ trong khoảng từ 0,0 đến 1, thì việc sử dụng trực tiếp chúng mà không cần tiện ích Độ mờ sẽ nhanh hơn nhiều.

Vẽ trực tiếp Hình ảnh hoặc Màu có độ mờ nhanh hơn so với sử dụng Độ mờ ở trên chúng vì Độ mờ có thể áp dụng độ mờ cho một nhóm tiện ích và do đó sẽ sử dụng bộ đệm ngoài màn hình tốn kém. Việc vẽ nội dung vào bộ đệm ngoài màn hình cũng có thể kích hoạt chuyển đổi mục tiêu kết xuất và việc chuyển đổi như vậy đặc biệt chậm trong các GPU cũ hơn.