さて、最後のレッスンでは、ヒーローアニメーションであるFlutterに付属する非常にシンプルですぐに使えるアニメーションを見て、実際にもっと多くのコードを必要とするものを実装するのがどれほど簡単かを見ました別のプラットフォームでそれを行いました。しかし、このレッスンでは、さらに一歩進んで、スーパーカスタムアニメーションの実装方法、画面上で発生させたいこと、背景色の変更またはサイズの変更について見ていきます。何かの移動や物事の移動。

しかし、時計のビートに合わせて、ループを繰り返すかどうか、実際にアニメーションで何をしたいのか、どれくらいの時間をかけたいのかを説明します。しかし、これを行うには、Flutterでカスタムアニメーションがどのように機能するかについてもう少し学ぶ必要があります。そして再び、スムージーの例えに戻ります。

この場合、Flutterを使用して構築するアニメーションのほとんどは3つのことだけが必要であり、3つの重要な概念を理解し、実際にそれがやりたいことを実行できるようにする必要があります。

そして、3つのコンポーネントは、ティッカー、アニメーションコントローラー、およびアニメーション値です。

今、ティッカーはまさにそのように聞こえます、それは時計のカチカチ音をたてるようなものです。

そのため、アニメーションを数えることができ、クロックを刻むたびにアニメーションの価値が変化し、形や色が変化するようにアニメーションを実行できるようにする必要があります。フリップブックにさまざまな絵を描くだけで作成できる最も単純なタイプのアニメーションについて考え、それをめくると、アニメーション効果が発生します。

それは、Flutterのアニメーションでもまったく同じです。

そのため、ティッカーがティックするたびに、新しいセット状態がトリガーされ、画面上で異なるものをレンダリングできます。これは、フリップブックのページをめくるようなものです。これで、他のコンポーネントはアニメーションコントローラーになります。これは、アニメーションのマネージャーのようなものです。

これらのすべては、アニメーションコントローラーを作成してそのプロパティを設定するときに、アニメーションの開始、停止、前進、ループバック、アニメーションの長さ、アニメーションの長さを決定するアニメーションコントローラークラスです。

そして最後に、アニメーションの値があり、これが実際にアニメーション化を行うものです。

したがって、通常、アニメーション値は0から1になります。

そして、その値を使用することで、コンポーネントの高さやサイズ、コンポーネントの色やアルファや不透明度などを変更できます。

それでは、実際のコードで試して、理論を実践に適用してみましょう。

したがって、私は現在welcome\_screen.dart内にいます。アニメーションを作成するために最初に行うことは、アニメーションコントローラーを構築することです。

最初に、AnimationControllerタイプの変数を作成します。そして、AnimatedControllerではなく、アニメーションコントローラーを作成したことを確認してください。

そして、コントローラーの名前を付けるだけです。

したがって、まだ何も初期化していません。

これは、AnimationController型のデータを保持する変数であることを言っているだけです。そして、実際に初期化したいのは、状態オブジェクトが初期化されるときです。

そのため、init stateメソッドをオーバーライドします。

したがって、このウェルカム画面の状態が作成された瞬間に、このメソッドが呼び出されることがわかります。

そして、アニメーションコントローラーを作成し、このコントローラープロパティに保存したいのはまさにその瞬間です。

アニメーションコントローラーのコンストラクターを使用して作成するため、アニメーションコントローラーについて説明します。これには、多くのプロパティが必要であることがわかります。

これらのプロパティの1つは継続時間です。このアニメーションをどのくらい継続しますか？

また、このプロパティは期間オブジェクトを取得するため、日、時間、分、秒などの期間を指定できます。

そして、私は私のアニメーションがちょうど1秒間続くと言うつもりです。

アニメーションコントローラーの必須プロパティである次のプロパティである期間を指定した後、アニメーションコントローラーに必要なパラメーターがあるため、この小さな黄色の三角形がまだここに表示されています。 「vsync」であり、まだ提供していません。

それでは先に進んでそれをしましょう。さて、vsync、これは正確には何ですか？これがティッカープロバイダーを提供する場所です。

それでは、アニメーションコントローラーのティッカーとして機能できるものは何ですか？

通常、ティッカープロバイダーは状態オブジェクトになるため、この場合は、ようこそ画面の状態になります。また、ようこそ画面の状態は、状態ウィジェットである親クラスを継承または拡張していることがわかります。そして、このウェルカム画面状態オブジェクトをティッカーとして機能できるものに変えるには、キーワード「with」を使用して、このクラスのウェルカム画面状態が単一のティッカープロバイダーとして機能できるように指定する必要があります。

つまり、ウェルカムスクリーンの状態を新しい能力で向上させているようなものです。単一のアニメーションのティッカーとして機能する機能。

したがって、複数のアニメーションがある場合は、TickerProviderStateMixinを使用します。

ただし、この場合は1つのものだけをアニメーション化するため、単一のチケットプロバイダーをアドオン機能として使用します。これを行うとき、基本的に使用しているのは、Mix inと呼ばれる概念です。そして、Mix inは、さまざまなタイプの機能を持つクラスを可能にします。

また、クラスから継承するのとは異なり、複数のミックスインを追加して多くの機能を追加できます。

コースの次のレッスンでは、Dartでのミックスインの仕組みとその正確な内容、およびクラスがティッカーとして機能するなどのことを実際に可能にする方法について詳しく説明します。

ただし、現時点で知っておく必要があるのは、このシングルテイクプロバイダーステートを追加することで、ウェルカムスクリーンステートをティッカープロバイダーとして機能させることです。

そのため、vsyncプロパティに移動し、ティッカーを探しているとき、ここで現在のウェルカムスクリーン状態オブジェクトをvsyncの値として提供します。

これがティッカーの予定です。

また、クラス独自のコードでクラスから作成されたオブジェクトを参照する場合は、キーワード「this」を使用します。したがって、このコード行は、だれが私のアニメーションコントローラーのティッカーを提供するかを示しています。

この現在のウェルカム画面状態オブジェクトになります。

これは、このクラスから作成されるまさにそのオブジェクトになります。これは、クラス宣言にそのミックスを追加したため、ティッカーとして機能するようになりました。

ティッカーができたので、コントローラーができました。

アニメーションを実際に使用して、アニメーションを開始するだけです。 init状態内の次の行でそれを行います。

コントローラーをタップして「進む」と言うので、これでアニメーションを進めます。現在、デフォルトでは、アニメーションコントローラーは数字をアニメーション化します。

そのため、ティッカーのティックごとに、その数が増加し、通常は0から1になります。

したがって、0.01から0.02になります。

したがって、1秒でティッカーで60ティックを取得する場合があります。この場合、コントローラーは60ステップで0から1にアニメーション化します。

そのため、コントローラーが何をしているかを確認したい場合は、リスナーを追加する必要があります。

したがって、controller.addListenerと言い、リスナーはコールバックを取得します。

そこで、ここにコールバックを追加します。

そして今、実際のアニメーションであるコントローラーの値を聞くことができますか？

それでは、controller.valueの値を出力しましょう。

コードをinit状態にすると、この状態が最初に作成されたときにのみ呼び出されることに注意してください。

そして、リロードしてもそれは起こりません。

ホットリスタートをクリックして、コンソール内を見て、印刷された値を確認しましょう。

しばらくの間、ゼロになっていることがわかり、0.32になっています。

0.36、0.39など

そして、一定の間隔でその値を刻み、その値を増やします。それで、この数は今や多くのことに使うことができますか？

たとえば、その番号を背景色に適用するとどうなりますか？

退屈な白い色ではなく、代わりに赤い色を使用して、不透明度を指定します。したがって、ある程度の不透明度を持つ赤色を指定します。

これで、不透明度の値は0〜1になり、1は完全に不透明、0は完全に透明になります。

したがって、これは1秒以内に0から1に移行することが既にわかっているコントローラー値に最適なアプリケーションです。

それを不透明度として提供しましょう。

そのため、controller.valueもdoubleであり、これはdoubleであるため、これで問題ありません。

したがって、ホットリスタートをもう一度押して画面をチェックすると、完全に透明になっていることがわかります。

そのため、その背後にあるもの、つまり黒のマテリアルアプリを表示するだけです。

しかし、実際には何も変わりませんか？

なぜそれが変化しないのか、なぜ自分の価値観が実際に変化してアニメーション化しているのを見ることができるのにアニメーション化しないのか？

もちろん、Flutterがどのように機能するかということに戻り、Flutterがcontrol.valueなどの値にどのように反応するかを思い出すと、この不透明度がアプリにあることをアプリに伝える必要があることを思い出してください汚くなるだろう。

そして、buildメソッドを呼び出して画面を再構築する必要があります。

つまり、addリスナーの内部では、set stateを呼び出すことができます。

また、値はアニメーションコントローラーで既に変更されているため、セット状態の内部で何もする必要はありません。

では、実行をヒットすると、アニメーションが正しく表示されるのを確認できますか？

基本的に背後にあるものを示している完全に透明な赤から、完全に不透明な赤になります。

そして、それを1秒間にわたってアニメーション化します。

アニメーションコントローラーに必要な唯一のプロパティはvsyncであり、ティッカーとして機能するものを指定します。

ただし、この場合、期間も指定しているため、アニメーションの実行時間も指定されています。

しかし、下限や上限など、変更可能な他のプロパティもあります。値を0から1にアニメートする代わりに、プロパティを変更してこれらの値を変更できます。

それでは、上限を1から100に変更しましょう。

そして今、背景色の容量を変更する代わりに、それを白に戻します。代わりに、テキストを変更しましょう。 「Flashチャット」と言う代わりに、0〜1の値を実際に出力させましょう。

それでは、文字列補間記号とドル記号を追加して、そのcontroller.valueをタップします。それがdoubleであるため、整数、整数に変更する必要があります。 0.0etcを見てください。次に、パーセント記号を文字列として追加するので、中括弧の外側に配置する必要があります。

そして今、アプリを実行すると、画面上にフラッシュチャットが表示される代わりに、ロードインジケーターが正しく表示されることがわかります。

読み込み機能とほとんど同じようにアニメーションします。0から100％になるとかなりきれいに見えます。

これは、コントローラーの値を使用して画面上の何かをアニメーション化できる別の方法です。

サイズをアニメーション化するために、この値を使用することもできます。それでは、このテキストをフラッシュチャットに戻して、ロゴコンテナに戻りましょう。また、高さを60にする代わりに、0から100に変更しましょう。

ここで、そのcontroller.valueを適用しましょう。

そして、すでにダブルを期待しているので、そのままにしておくことができます。

そのため、アプリを実行してアニメーションが表示されると、ロゴが1秒間に0の高さから100ピクセルの高さまでアニメーションで表示されるはずです。

そのため、そのコントローラーの値を使用してあらゆるものをアニメーション化することは非常に柔軟であることがわかります。

現時点では、アニメーションは直線的に発生します。そのため、サイズが大きくなるにつれて、直線的にサイズが大きくなります。

そして、コントローラーの値を見ると、増分が0から100までかなり均等になっていることがわかりますか？

そして、これは1秒間に約60回行われます。

さて、これを少し違うものにしたい場合はどうしますか？それでは、Curved Animationというクラスを使用できます。

この方法で、カーブに沿ってアニメーション値を変更できます。また、使用できるカーブのタイプには、バウンスなどが含まれます。また、Flutterには、グラフ上で各タイプの曲線がどのように見えるかを示す素晴らしいドキュメントがあり、0から1に進み、たとえば1秒に沿って進みます。そして、これらの各曲線を適用した結果としてのアニメーションが、変更しようとしているものに応じてどのように見えるかを確認できます。

それでは、本当に簡単なことを試してみましょう。減速曲線を使用して、コードでそれをどのように行うかを見てみましょう。

そのため、曲線を使用するには、別の変数を作成する必要があり、この変数はAnimation型になります。そして、アニメーションを単にアニメーションと呼び、初期状態の中で、そのアニメーションを新しい曲線アニメーションに初期化します。また、曲線アニメーションには2つの必須パラメーターが必要です。

1つは親です。

そして、親がアニメーションコントローラーでなければならないことに注意してください。

これで、親がこの曲線を適用する対象になります。この場合、これがコントローラーになります。したがって、既存のアニメーションはこのコントローラーから取得されます。

2番目のプロパティは、アニメーションにどのような曲線を適用するかです。

以前は、曲線から1つを選択しました。これは減速曲線です。

それがこのように見えるものです。

そのため、スケールである場合、最初は高速であり、最後にスケールアップが減速します。

カーブアニメーションを作成し、このカーブをどのアニメーションに適用し、どのタイプのカーブを適用するかを指定したら、最後に確認する必要があるのは、コントローラーに曲線アニメーションを適用しているため、実際には1を超える上限を持つことはできません。

これらの曲線は、0から1まで描画する必要があります。

したがって、この上限をそのままにしておくと、実際に例外がスローされ、アプリまたはクラッシュが発生します。

まず、どのように見えるかをお見せしましょう。

コントローラーの値を使用する代わりに、アニメーションの値を使用します。これは、アニメーションがコントロールアニメーションの上に適用されるレイヤーにほとんど似ているためです。

それでは、animation.valueを出力し、heightプロパティ内で、それをanimation.valueに変更して、選択した曲線に基づいて生成されるこの新しい値を使用しましょう。

そのため、実行をヒットすると、アプリがクラッシュします。

そして、上限を1より大きくすることはできず、1以下でなければならないためです。

したがって、曲線を使用して、上限が1より大きくないことを確認および確認する場合は、常に覚えておいてください。そして、上限として必要な場合は、実際に何も変更する必要はありません。

しかし、画像のサイズに合わせて0から1のアニメーションを作成することはあまり役に立ちません。

0と1の間の変化は見られません。

したがって、その数値に100を掛けて、効果を誇張しましょう。

そして、アニメーションの値をそのまま印刷して、どのように見えるかを確認します。

それでは、減速曲線を適用した場合のアニメーションの様子を次に示します。最初は非常に大量に増加し始めるため、15から23に進み、29から33になりますが、2番目の終わりに向かって、非常にゆっくりと増加し始めます。

そしてその効果は、画像のサイズが非常に急速に大きくなり、その後、減速して最後に向かって拡大を停止することです。

ここで、他の曲線がどのように見えるかを自由に確認してください。

たとえば、曲線の1つとして「イーズイン」を設定できますが、実際にはこの曲線は反対のようなものになります。

開始するのは遅くなりますが、最後には少し速くなります。アニメーションを逆にしたい場合、大から小にしたい場合はどうなりますか？

コントローラーを前進させる代わりに、実際にコントローラーを逆にすることができます。そして、アニメーションを反転する必要があると言えます。反転するときは、fromプロパティを指定することをお勧めします。

これが新しい出発点になります。

したがって、100の1倍で開始し、その後0まで下がったとしましょう。

実行を開始してアニメーションを見ると、最初は大きくなりますが、再び100から1に戻って小さくなります。アニメーションをループさせたい場合、大きくしたい場合その後、小さくて大きくなり、小さくなりますか？

それを行うためには、アニメーションが実際に完了したとき、リバースアニメーションが完了したとき、フォワードアニメーションが完了したときを知る必要があります。

それでは、先に進み、これを前に戻しましょう。

そして、addStatusListenerと呼ばれるアニメーションにアタッチできるメソッドを使用します。

そしてここで、アニメーションのステータスをリッスンするコールバックを渡すことができます。

それを印刷して、ステータスが実際に何であるかを見てみましょう。

そして、アニメーションと一緒にコンソールを見ると、アニメーションが完了すると、アニメーションのステータスが完了したことがわかります。

したがって、順方向のアニメーションを実行している場合、.completeのステータスを確認することで、実際にアニメーション化が完了したことを検出できます。

ただし、リバースアニメーションを使用する場合は、1から0に変更します。

この場合、取得されるのは、アニメーションステータスが却下されるということです。

そのため、反転したアニメーションの終わりは閉じられ、順方向のアニメーションの終わりは完了します。

つまり、ステータスがAnimationStatus.completedに等しいかどうかを非常に簡単に確認できるようになったということです。つまり、前方アニメーションが完了し、後方に移動できることを意味します。

それで、1からcontroller.reverseと言うことができます。そして、ステータスがAnimationStatus.dismissedと等しいかどうかを確認するか、そうでなければ、それがその逆アニメーションの終わりであり、それを再び前進させることができます。そのため、まずコントローラに前方へのアニメーション化を指示することから始めます。そのため、小さなものから大きなものへと進みます。

それが完了すると、ステータスが完了としてトリガーされ、逆になります。

そして、リバースアニメーションが完了すると、ステータスがdismissedになり、再び前進します。

だから今実行すると、基本的に永遠まで私のアニメーションが大から小、大から小に跳ね返るのがわかります。

そして、これは、実際にアニメーションコントローラーを破壊しない限り、このアニメーションが永遠に続くことを言及する良い点です。現時点では、この画面が閉じられたとしても、そのコントローラーはまだ存続しており、リソースを消費しています。

したがって、アニメーションコントローラーを使用するときはいつでも、disposeメソッドをタップしてオーバーライドすることが非常に重要です。これにより、画面が破棄されるとき、またはこのようこそ画面の状態が破棄されるときに、コントローラーも廃棄します。

したがって、この方法では、メモリ内にとどまり、すべてのリソースを占有することにはなりません。

それで、私があなたに見せたいアニメーションの最後のタイプは、トゥイーンアニメーションと呼ばれるものです。

基本的には、定義済みのトゥイーンアニメーションのセットであり、本質的に値の間に入ります。

たとえば、開始色と終了色がある場合、トゥイーンはスムーズな移行で最初から最後まで行きます。

それではまず、addStatusListenerを削除しましょう。

アニメーションが前後にバウンドすることはもうありません。

また、カーブしたアニメーションも削除しますが、そのアニメーション変数は保持しますが、代わりに別の値を持つように署名し、代わりにColorTweenアニメーションに設定します。

このカラートゥイーンアニメーションは、開始と終了の2つのプロパティのみを取ります。

開始点は開始色なので、colors.redを選択し、終了点をcolors.blueに変更できるとしましょう。これは本当にいように見えますが、それでも私たちにとってはうまく機能します。

そこで、ここでカラートゥイーンを作成し、トゥイーンの定義が完了したら、その上でアニメーションを呼び出して、このカラートゥイーンを使用するアニメーションを返します。

そして、それがここでこのアニメーション変数に保存するものです。そして、このカラートゥイーンをアニメーションコントローラーに適用します。そこで、赤から青に移行するカラートゥイーンを作成し、そのアニメーションをコントローラーに適用して、まったく新しいアニメーションを作成します。

これでアニメーションを使用できるようになったので、この高さを60に戻し、アニメーション化しないようにします。

私はロゴが跳ね回るのが好きではありません。代わりに、ここでその色を適用します。

そこで、背景色をanimation.valueに変更します。そして今、アプリを実行すると、1秒以内に色が赤から青に変わることがわかります。

そのため、見やすくするために、おそらく1秒より少し長くすることができます。

これを3秒以上に変更すると、赤から青に移行するときに遷移が少し遅くなることがわかります。

そして、コンソールでは、animation.valueをログに記録しているため、実際には赤から青までの各ステップを通過する色を構築していることがわかります。

そして、赤から青に行くために、変化の小さな増分を構築します。

これがトゥイーンまたはインビトウィーンアニメーションです。

ボーダー半径トゥイーンや整列トゥイーンなど、使用できるその他のトゥイーンアニメーションがたくさんあります。

しかし、それらはすべて開始値と終了値を取り、トゥイーンを作成するのと同じ方法でコードを構築し、アニメーション化してアニメーションコントローラーに適用することができます。そして、そのアニメーションの値をビルドメソッド内で使用します。

したがって、カラートゥイーンのanimation.valueは明らかに色になりますが、境界線半径トゥイーンのアニメーションは境界線半径になり、ボタンを言うために適用できます。

このレッスンでは、さまざまな種類のアニメーションを見てきましたが、実際に保持するのは、おそらくグレーまたはブルーグレーで始まるカラートゥイーンアニメーションです色、そして白色の背景で終わるので、colors.whiteにします。そして、それを1秒に戻します。

このように、アプリが最初に読み込まれ、ユーザーにようこそ画面が表示されると、ようこそ画面が表示されるだけの簡単なアニメーションになります。次のレッスンでは、Dartのミックスインについて深く掘り下げ、この「with」キーワードが何をし、これらのミックスインがクラスに対して実際に何をするのかを学びます。

したがって、ミックスインについて多くのことをすでに知っていて、このコードを理解するのに問題がない場合は、次のレッスンをスキップしてください。しかし、これが何であるか、どのように機能し、なぜそれを使用するのかを知りたい場合は、次のレッスンでお会いしましょう。