このレッスンでは、ウィジェットを個別のクラスとして抽出およびリファクタリングする方法を主に見ていきます。

そして、これを行う理由は、アプリがますます複雑になるにつれて、ウィジェットツリー内で繰り返されるアプリケーションの一部を取得して削除し、それらを単一のウィジェットに統合できるようにするためです。

そして、同じ外観または同じ機能が必要な場合はいつでも、そのウィジェットを使用できます。

しかし、これを行う前に、まず実際に何かを画面に表示する必要があります。

ユーザーインターフェイスを設計する必要があります。そして、これを作成する最初のステップは、実際にスケルトンを作成することです。

そのため、画面上に表示される個々のカードの色はこのようにわずかに明るいため、個々のコンポーネントまたはエリアにグループ化できます。また、ユーザーは、BMIを計算するために集まったさまざまな部分を見ることができます。

したがって、コンテンツまたはテキストを実際に追加する前に、まず、これらの各領域の背景となるカードを作成します。

そのため、私たちは少しこのようなものを目指しています。

まず、これらのカードの1つを作成することから始めましょう。実際にはテキストは必要ないので、足場の本体にあるすべてのものを削除します。

代わりに、ここで欲しいのはコンテナです。

これで、コンテナに背景色が設定され、このデザインで表示される色になります。

そして、すでに16進コードを抽出しているので、単純に追加したい場合は0xFFの色で、1D1E33です。

そして、スクリーン上に、少し明るい紫の色のコンテナができているはずです。

今、私はそれが画面全体をカバーしたくないので、多分それに高さを与えられるようにしたい。

たとえば、高さ200、幅170を指定します。

これで、左上に貼り付けられた小さな正方形ができました。しかし、実際には、このように互いに分離されるように、すべてから少しのマージンを持たせたいと思っています。

そこで、コンテナにもマージンを追加します。

そして、私はすべての側面から約15のマージンを持っていることはかなり良いことだと考えました。

それは私たちが今持っているものです。

次のステップは、コンテナの端が非常に鋭いことです。

このように少し丸みを帯びていれば、私はそれが大好きです。

それはもう少しフレンドリーに見えます、そしてそれは非常に素晴らしいに見えます。少なくとも私の目には。これを実現するには、コンテナに境界線の半径を作成できる必要があります。

これは、Webを実行した場合にsebsitesで表示される境界半径に非常に似ています。

開発。

そして、それを実装するために、装飾プロパティと呼ばれるものを追加します。

また、装飾プロパティには、boxDecorationと呼ばれるものが必要です。また、ボックスの装飾には、色、画像の境界線、境界線の半径などのプロパティがあります。

これにより、角を丸くするようにボックスを変更できます。

そこで、その境界半径を挿入し、半径10ピクセルの円形の境界半径として設定します。

保存してみましょう。アプリが実際にクラッシュすることがわかります。

理由は、コンテナにcolorプロパティを使用するとき、実際にはボックス装飾のcolorプロパティの単なる省略形だからです。

ボックスの装飾にも色のプロパティがあり、それが実際にコンテナの背景の色を設定する適切な方法であることに注意してください。

しかし、非常に頻繁に使用されるため、装飾、ボックス装飾、色などを常に書く必要はありません。 Flutterチームは、このプロパティを使用して、コンテナに簡単な色を追加することを簡単にしました。

しかし、ボックス装飾がある場合は、色をボックス装飾に移動する必要があります。

これにより、コードがクラッシュしなくなり、角が丸くなった明るい紫色の背景ができます。

ここで、アプリを調べて、これらの各カードにハードコードされたサイズを追加すると、実際には画面のサイズに適応しません。

たとえば、画面が横向きに変更された場合、または画面サイズが小さい場合、画面はオーバーフローし、ウィジェットが画面外にある黄色のバーが表示されます。

だから、それは私たちが望むものではありません。その解決策は、以前に見た拡張ウィジェットを使用することです。

先に進み、高さと幅を削除しましょう。そして課題は、今作成したコンテナを使用してこのレイアウトを作成することですが、使用可能なサイズに合わせて画面上にこのようにレイアウトします。

ビデオを一時停止し、この外観を実現できるかどうかを確認します。

大丈夫。

そのため、最初に気付くのは、すべてが列に積み上げられていることです。そして、最初の列の中には、2枚のカードの行があり、最後の列にも同じことが当てはまります。

それでは、容器を本体から切り取り、ここに置きましょう。次に、子を含む列を追加します。そして、このコラムには3つの拡張ウィジェットが含まれ、それぞれに独自の子があります。

これを単純にコピーして3回貼り付けます。したがって、これらの展開された各ウィジェットは、ここの列の1行、2行、3行になります。

これで、最初の展開されたウィジェットには、行ウィジェットである子が作成されます。また、いくつかの子があり、2つの拡張されたウィジェットがあります。

これで、それぞれが拡張されたウィジェットである3行の列ができました。

最初の行に行ウィジェットを追加して、2つの拡張ウィジェットを使用できるようにし、最後の行にそれを複製します。

したがって、この拡張されたウィジェット全体を単純にコピーし、最後の行をそれで置き換えます。

現時点では、保存して画面を見ると、実際には何も起こりません。また、ここにコードが横たわっているだけなので、それほど多くのことはしません。しかし、先に進んでコンテナをコピーし、展開されたウィジェットのすべてに子として追加すると、展開されたウィジェット、列、および行を使用して画面を実際にレイアウトするすべてのハードワークが実際に表示されます画面に実装され、前に見たデザインを真似しようとしています。混乱を招く場合は、列、行拡張ウィジェット、フレックスプロパティなどについて説明したビデオに戻ってください。そうすれば、これが本当に明確になります。

すべてのウィジェットの最後にコンマを追加して、保存時にDartがコードを適切に再フォーマットして、すべてが整然と見えるようにします。したがって、このコードは機能し、私たちが望むものとほとんど同じように見えますが、非常に反復的ですよね？

入れているコンテナが完全に同一であっても、実際にコンテナを6つの場所にコピーして貼り付けました。

これは本当にプログラマーのスパイダーセンスを引き立たせるはずです。繰り返しの量を減らすことができるようにコードを実際にリファクタリングする良い機会です。これらのコンテナを抽出するために、実際にはFlutterからアクセスできる本当に巧妙なトリックがあります。

繰り返されるウィジェット（この場合はコンテナー）を選択し、[フラッターアウトライン]タブに移動すると、ウィジェットツリーで選択されていることがわかります。

ここで右クリックすると、抽出ウィジェットを選択できます。

これにより、新しいウィジェットの名前を尋ねられ、再利用可能なカードを呼び出して、これを新しいクラスにリファクタリングします。

したがって、これは新しい再利用可能なカードであり、Flutterは実際にこのコードをすべて作成しました。このコードは、ステートレスウィジェットを拡張する個別のクラスにウィジェットを抽出するだけです。マージン、装飾、色など。それが、再利用可能なカードと呼ばれる理由です。

Flutterがこのコードを自動的に生成すると、「新しい」キーワードが作成されます。

そして、私たちは今のところそれを見たことはありません。

そしてその理由は、Dart 2.0の最新バージョンであるDartでは、これは実際には廃止されたためです。

そのため、最新バージョンのDartを使用してクラスから新しいオブジェクトを作成する場合、実際にはこの新しいキーワードはまったく必要ありません。クラスの名前を単に書いてから括弧を追加するだけです。

同じコンテナがある場所をすべて削除して、代わりに新しい再利用可能なカードを作成することができます。これで、コードが大幅に簡素化され、コードが30行ほど減ったことがわかります。

そして今、それは非常に明確に見えます。

したがって、再利用可能なカードに問題が発生した場合、6つの異なる場所で修正するのではなく、作成された場所を見つけて実際にここで対処するだけです。ステートレスウィジェットでは、このコンストラクターがここにあることがわかります。

また、フォーマットの方法を変更すると、Dartクラスとオブジェクトについて最初に学んだときに、Quizzlerモジュールで学んだことをなじみ始めます。

この場合、キーと呼ばれるこのことについて話し続けます。

では、キーとは正確には何ですか？キークラスは、ウィジェットの状態を追跡する必要があるときに使用するものです。

また、これは、ウィジェットがウィジェットツリー内を移動しているとき、ウィジェットツリー内でウィジェットの位置を変更しているときに特に役立ちます。

これは通常、ウィジェットがアニメーションなどで画面上を物理的に移動しているとき、または画面上でスクロールしているものを見ているときに発生します。

しかし、他のほとんどの場合、実際には必要ありません。

そこで、自動的に生成されたこのコンストラクターを削除して、独自のコンストラクターを作成します。

これらのキーを実際に使用するタイミングとその用途について詳しく知りたい場合は、このリンクに移動して、フラッターチームのエミリーが実際に使用方法とその用途を詳しく説明します。その場合、ウィジェットがその状態を追跡できるようになります。

これへのリンクもコースリソースにあります。

ただし、この場合、キーを使用して何もする必要はありません。しかし、再利用可能なカードで使用するプロパティの1つにカスタム値を指定できるようにするには、コンストラクターが必要です。

再利用可能なカードについて変更する可能性のあるプロパティは、実際にはカードの色です。これは、常に同じ色にしたくない場合があるためです。

最終結果を見ると、男性または女性のセレクターをクリックすると、実際にカードの背景色が変わり、実際に選択されていることと非アクティブであることが表示されます。

これは、再利用可能なカードについて変更する必要があるものです。したがって、新しい再利用可能なカードウィジェットを作成するときに、それを再利用可能なカードのプロパティとして渡すことは理にかなっています。

したがって、先に進んでプロパティを作成できます。そのプロパティは色タイプになります。

そして、私は単にそれを色とも呼びます。

そして、私はロンドン出身なので、このように綴りますが、好きなように綴ることができます。しかし、実際には、私が作成したこの色プロパティを、コンテナウィジェットの色プロパティと区別するのに役立ちます。

だから私はこのように綴ることを好む。

そして、値なしで開始するので、単純にセミコロンで終了します。次に、再利用可能なカードのコンストラクターを作成します。コンストラクターの作成方法を簡単に思い出させるために、クラスの名前を追加し、括弧のセットを開き、ここに初期化するプロパティ（この場合はthis.colour）を追加します。その後、そのステートメントを終了します。

このプロパティの名前を参照できるようにしたい場合は、名前を付けたいすべてのプロパティの周りに中括弧のセットを追加することもできます。

ですから、ここで、再利用可能なカードを作成するときに、実際にcolorプロパティを追加して、カスタムカラーを指定できます。

たとえば、これは多分これ。

また、新しい再利用可能なカードウィジェットを作成するときに常に同じ色を使用する代わりに、私の再利用可能なカード内で、このウィジェットの作成時に渡された色を使用できます。

このように、カスタム機能やカスタムルックアンドフィールが必要な場合はいつでも、カスタムウィジェットの作成に近づいています。現時点では、このプロパティは実際にはオプションのプロパティです。

ここで確認できますが、ここで色を指定した再利用可能なカードを作成しました。

しかし、ここでは、コンストラクターで何も指定せずに、再利用可能なカードウィジェットを実際に初期化しました。

ここで警告やエラーが表示されないため、これは許可されています。

しかし、考えてみると、すべてのカードに背景色が必要になるでしょう。

null色を使用することはできません。

したがって、再利用可能なカードウィジェットを作成するときは常に色を要求するのが理にかなっています。特定のプロパティを必須にするために、コンストラクタで必要なキーワードである注釈を追加できます。そして、スペースの後に、必要なプロパティがあります。

上にスクロールすると、これらの他のすべての再利用可能なカードに黄色のハイライトが表示されます。

そして、この再利用可能なコードのパラメーターの色が必要であることを教えてくれます。

だから入れなければならない。

それでは、すべての再利用可能なカードに同じ色を追加しましょう。

そして今、右側の警告は消えます。

そのため、保存してアプリを見てみると、まったく何も変わっていません。しかし、何かがうまくいかなくなったとき、より同期され、読みやすく、修正がはるかに簡単になるようにコードを修正しました。

そして、再利用可能なカードである独自のカスタムウィジェットを基本的に作成しました。このウィジェットは、新しい再利用可能なカードを作成するときに指定するマージンと背景色を備えた角丸コンテナである特定のウィジェットが必要なときにいつでも使用できます。さらに、ウィジェットにプロパティを追加すると、デフォルトのFlutterウィジェットに非常に似たウィジェットの構築が開始されることを想像できます。たとえば、コンテナにはmarginプロパティ、decorationプロパティ、colorプロパティがあります。

そして、これらはすべてその特定のクラスにあるプロパティですか？

コンテナはステートレスウィジェットであり、配置、パディング、装飾などのプロパティがあることがわかります。

そして、新しいコンテナウィジェットを作成するときにこれらすべてを指定できます。

ただし、この場合、新しい再利用可能なカードウィジェットを作成するとき、実際に独自のコードを調べて、どのように見えるか、どのように動作するかを確認しています。

たとえば、単純に青色のカードを作成できます。

そして、プロパティを変更するとフローティングアクションボタンまたはフラットボタンが変更されるのと同じように、ウィジェットの外観が更新されます。

ほぼ同じですが、私たちが定義しています。

さて、最後に見せたいのはここです。実際に警告があります。

そして、その上にカーソルを置くと、このクラスが不変としてマークされていることがわかります。しかし、そのインスタンスフィールドの1つ以上は最終的なものではありません。つまり、ここでこれを参照する再利用可能なCard.colorフィールドです。

そして、それが不満を抱いているのは、このフィールドまたはプロパティの前にfinalというキーワードがないことです。

これが重要である理由と、これらの警告が表示される理由は、finalがこのプロパティを不変にするためです。

つまり、新しい再利用可能なカードを作成するときに、色を設定でき、それを再び変更することはできません。

これがなぜ重要で、実際にどのように機能するのか、

Dartプログラミング言語でのfinalとconstについて、そしてこれがウィジェットの動作と画面上での動作にどのように影響するかについて、次のレッスンまでお話しします。

それで、そのすべてについて、次のレッスンでお会いしましょう。