大丈夫。

これで、すべてのユーザーインターフェイスがほぼ完成したので、アプリに機能を追加する準備が整いました。

そして、あなたは現在、私たちがまだ取り除いていないという1つの警告があることに気付くでしょう。

そして、それはチェックボックスに関連しています。

そして、シミュレーターまたはエミュレーター内で使用しようとすると、まったく何も起こらないことに注意してください。

まだ機能がありません。

チェックボックスを機能させる方法を理解するために、まず状態についてもう少し理解する必要があります。

それでは、状態とは正確には何ですか？

それは、ユーザーインターフェイスを構成するすべての変数の値と考えることができます。

たとえば、現在のTodoeyアプリには、空のチェックボックスがたくさんあります。これらのチェックボックスにはvalueというプロパティがあり、その値によってチェックボックスに表示されるものが決まります。

したがって、falseに等しい場合、空のチェックボックスが表示されます。

しかし、それがtrueに等しい場合、チェックボックスが表示されます。

しかし、これは非常に手作業ですよね？

では、どうすれば自動的にこれを行うことができますか？

その場合、変更可能な変数が必要になります。

isCheckedという変数を作成し、デフォルトでfalseに設定し、これらの各チェックボックスを設定して、この変数をチェックボックスの状態として使用するとします。 isCheckedがfalseの場合、すべてのチェックボックスは空です。しかし、ユーザーがチェックボックスの1つをタップし、その変数の値をtrueに変更すると、ユーザーインターフェイスが更新され、チェックボックスの更新された状態に基づいてアプリの最新バージョンが表示されます。

現在、一般的に2種類の状態を考える傾向があります。

ローカル状態があります。たとえば、ページネーションを許可するアプリがある場合、ユーザーが現在表示しているページなどの有用な情報ですが、通常はアプリ内の他の場所で必要な情報ではありません。

そのため、アプリ全体で使用される値を保存する必要があるのは、データや変数ではありません。

ローカル状態では、セット状態を使用して変数の値を切り替えるのは非常に簡単です。

この場合、ユーザーがチェックボックスの1つをタップし、set stateを使用してisCheckedをfalseからtrueに変更するとします。

その後、それが発生するとすぐにチェックボックスが再描画され、チェックボックスの状態が変更されたため小さなチェックマークが表示されます。

新しいものに設定しました。

これで、チェックが前後に変わることでこれを切り替えることができます。

しかし、この場合、isCheckedのこの値を保存して、アプリ内の他の場所で使用することはしません。

そのため、現在、チェックボックスには多くのコードが含まれていません。

これは非常に単純なチェックボックスですが、1つのプロパティを追加してみましょう。

アプリの他の部分と一致するように、アクティブな色を水色のアクセントに変更しましょう。次に、このチェックボックスを別のウィジェットとして抽出します。

そこで、これをTaskCheckboxと呼び、沼地の標準チェックボックスと区別し、抽出したので、実際にコンストラクターを削除し、ステートレスウィジェットからステートレスウィジェットに変換します。ステートフルウィジェット。このチェックボックス内で設定状態を使用して、その状態を更新できるようになりました。そして、その状態は、チェックボックスの値をfalseにする必要があるかどうか、またはtrueにする必要があるかどうかです。そして、その値がユーザー入力に基づいていることを望みます。

したがって、この場合、チェックボックスにはonChangedというプロパティがあり、このコールバックにチェックボックスの新しい値を送信し、isCheckedと呼ばれるブール変数を実際に作成できます。

そして、デフォルトで、実際にfalseに設定し、ユーザーが実際に変更するときにここで下に移動することができます。つまり、falseからtrueに切り替えることを意味し、isCheckedを新しい値に等しくなるように変更します。

したがって、チェックボックスの状態を変更して、この変数isCheckedを使用できます。

そして最後に、このisChecked変数をユーザーが選択した新しい値に変更してからbuildメソッドを呼び出すことができるように、私たちが知っている、大好きなセット状態を追加します。そのようにして、チェックボックスの状態の最新の値を使用して、新しいCheckBoxを構築します。

そのため、保存ボタンをクリックすると、各チェックボックスを個別にオンまたはオフにすることができます。同じステートフルウィジェット内でラップされているため、非常にローカルな状態です。ウィジェットツリーでは非常に低いため、再構築のコストはそれほど高くなく、isCheckedの値は他の場所に移動していません。では、グローバルステートはどうでしょうか。

グローバルステートを管理しているときは、基本的にいくつかの変数の値を保持しており、アプリのさまざまな部分でそれを使用しています。

たとえば、アプリを非常にシンプルなウィジェットツリーとして表すことができます。

一番上にリスト画面があり、そこからアバター画面とタスクリストに分岐します。

タスクリストにはタスクタイルが配置され、各タイルにはチェックボックスとタスクのタイトルであるテキストがあります。

チェックボックスウィジェット内でその情報のみが必要な場合は、ステートフルウィジェットを使用し、状態が変更されるたびにset stateを使用してチェックボックスを再構築できます。

しかし、アプリ内のどこかでisCheckedの値が必要な場合、たとえば、チェックオフしたタスクのタイトルに取り消し線を入れて、チェックボックスだけでなくその状態も更新したい場合などウィジェットは、同じ変数、つまりこのisChecked変数の値を使用して状態を更新します。

それでは、どうすればこれを達成できますか？この変数isCheckedをタスクタイルウィジェットのレベルを1つ上に移動することから始めましょう。

ここにスペースを作りましょう。それからここにドラッグします。これで、タスクタイルウィジェットは現在ステートレスウィジェットであるため、警告が表示されていることがわかります。

それをステートフルウィジェットに変換して、実際に変化するプロパティを実際に持つことができるようにします。

このisCheckedプロパティを実際にここに移動したので、チェックボックスにいくつかのエラーが表示されています。レンダリングするチェックボックスの種類と作成するテキストウィジェットの種類を管理するステートフルウィジェットがある場合、このチェックボックスウィジェットをステートフルウィジェットにする必要はありません。実際にステートレスウィジェットに変換し、比較的シンプルに保つことができます。

そこで、最後にブール値になるブール値を追加して、定数のままにしますが、ここでisChecked値を変更するたびに、このブール値を使用してチェックボックスを再描画して再構築します。 。

そこで、これをcheckboxStateと呼び、this checkboxStateを使用してTaskCheckboxを初期化します。

涼しい。

これで、以前にisCheckedを使用していたすべての場所を置き換えて置き換えることができますが、このセットステートパーツは今のところ残しておきます。ステートレスウィジェットが作成されたため、ステートがないため、ステートレスウィジェットの状態を設定できません。

すぐにそれを整理しますが、今のところ、このTaskCheckboxをより暗いウィジェットに変更し、必要なのは、初期化されたときに渡されるcheckboxStateの値を使用して自分自身を再構築することです。ステートフルウィジェットとステートレスウィジェットの違いを覚えているのであれば、このようなものです。

したがって、ステートフルウィジェットは、変更できるものと考えることができ、それらの変数の値に応じて状態を変えることができます。

だから最近、私はたくさんの本当に中毒性のあるGIFに出会いました。そこにはあらゆる種類のテーブルがあり、それらは崩壊して拡大しているようで、さまざまなクレイジーな方法でそれを行います。

そして、それをステートフルテーブルと考えることができますか？

それは変更することができ、更新することができ、私たちが望むものに応じてすぐに変更することができます。

ただし、ステートレスウィジェットは、通常のテーブルのようなものです。

そして、両者の違いは、このテーブルのコストが$ 2,000であるのに対して、このテーブルのコストは$ 200であるということです。

そして、その理由は、このステートフルテーブルがより多くの材料を必要とするためです。それは生産するのにより多くの費用がかかり、それを構築するのに多くの労力がかかります。

ウィジェットでも同じです。

より正確には、実際に変化しているのはステートフルウィジェットではなく、実際には状態オブジェクトです。オブジェクトをステートレスウィジェットとして使用する場合は（そのままの状態で）、変更する必要がある場合は、単にこのウィジェットを削除して新しいウィジェットを作成します。

しかし、状態オブジェクトがある場合、状態オブジェクトがあり、その状態は変更できます。

つまり、ユーザーインターフェイスを更新するために変数を変更するため、通常、状態内の変数は最終としてマークされません。

そして、状態オブジェクトのプロパティの変更が完了したら、そのコピーを作成します。これがステートフルウィジェットです。このように、ステートからステートフルウィジェットを作成する以外に、非常に小さなコードを含むステートフルウィジェットクラスがあることがわかります。そして、ここには実際の状態オブジェクトであるTaskTileStateがあります。

そして、これは、値を変更できる変数がある場所です。そのため、最終変数でマークされていません。

そして、変更を使用してbuildメソッドでユーザーインターフェイスを更新しますが、ステートレスウィジェットでは、すべてのプロパティが最終的なものである必要があります。

これらのプロパティを変更することはできませんが、これを更新する必要がある場合は、このTaskCheckboxの真新しいバージョンを初期化し、最後のものを破棄します。

ここで、TaskCheckboxイニシャライザーが1つの引数を必要としていることに注目してください。これが状態です。したがって、ここに進み、そのisChecked値を渡すことができます。

trueまたはfalseであり、チェックボックスに渡されて、ティックまたは空のボックスを表示できるようになります。状態をTaskCheckboxの親まで持ち上げたので、テキストウィジェット内でこのプロパティにアクセスできるだけでなく、タスクチェックボックスに渡すこともできます。

つまり、テキストウィジェットにテキストスタイルを追加するなどのことができ、追加したいのは装飾です。

タスクがオフになっているときにテキストを消すことができるようにしたいので、使用する装飾はTextDecoration.lineThroughです。

そして今すぐアプリを再実行する場合、ステートフルウィジェットをステートレスウィジェットに切り替えるか、その逆を行うたびに、最初から実際に再実行する必要があります。ホットリロードでは不十分です。

これで、3つのすべてのテキストウィジェットに線が引かれていることがわかりますが、これは明らかに、タスクアプリにしたい方法ではありません。

装飾がこのプロパティの値に依存するようにしたいので、ここで必要になります。

したがって、ここに単に装飾を追加するのではなく、isCheckedがtrueであるかどうかを確認するためにチェックし、そうであれば、装飾を通してその行を追加します。

しかし、そうでない場合は、装飾をnullまたは装飾なしに設定します。

だから今、保存を押したら。

しかし、このプロパティをtrueに変更すると、すべてのタスクが消され、すべてのチェックボックスがチェックされます。

そのため、このプロパティは解除され、状態はすべてのダウンストリームウィジェット（テキストとタスクチェックボックス）で使用されています。

だから、それはすべて非常によくて良いことです。

それをすぐにfalseに復元しましょう。

しかし問題は、このユーザーインタラクションを親ウィジェットにどのように戻すかということです。ステートを上げたので、変更できないステートレスタスクのチェックボックスができたため、独自のステートを設定できません。

それでは、ユーザーインタラクションをタスクタイル状態に戻すにはどうすればよいでしょうか？私はまだこれらのチェックボックスを正しく動作させたいのですか？

しかし、現時点では何も変わっていません。

そのため、このデータを親に戻す方法の1つは、コールバックを使用することです。

そして、ほぼ左、右、中央のコールバックを使用しており、実際、チェックボックスのonChangedプロパティまたはボタンのonTapなどのユーザーインタラクションがあるたびに、常にコールバックをそこに入れていますユーザーの操作を待ってから、その機能をトリガーします

ユーザーがその特定のアクションを実行した後にのみ、たとえばチェックボックスを変更しました。

したがって、コールバックを作成する場合は、このonChangedプロパティにあるすべてのコードを取得するだけで済み、タスクタイル状態に移行できます。

それで、checkboxCallbackと呼ばれるトグルチェックボックスと呼ばれる新しい関数を作成します。

そして、今コピーしたものを単に貼り付けて、まだコメントされている部分のコメントを外します。

より正確に言うと、checkboxCallbackを取得したので、実際にはブール型のデータ型を取り、ここで新しい値を呼び出しました。これは、チェックボックスの新しい値だったからかもしれませんが、それをチェックボックス状態または現在のチェックボックス状態と呼ぶことができます。それが本質的にそうなるからです。

ここでonChangedでトリガーされ、チェックボックスが変更されると、このコールバックを呼び出し、新しいcheckboxStateを渡し、その値を使用してブール値isCheckedを更新します。

そのため、checkboxCallbackを作成したので、それを使用する必要があり、作成時にTaskCheckboxに渡すことで使用します。

したがって、関数型になる最終プロパティを追加し、toggleCheck boxStateと呼びます。そして、TaskCheckboxのコンストラクター内に追加して、新しいタスクチェックボックスを作成するときに関数を渡し、それをここに保存して、onChangedで使用できるようにします。

そのため、この関数をonChangedでここに渡します。

2つの引数が必要なため、ここで少し赤い波線が表示されます。2番目の引数は、チェックボックスが変更されたときにコールバックとして使用される関数です。

したがって、関数の名前はcheckboxCallbackであり、保存をクリックすると、エラーが消えます。

このTaskCheckboxコンストラクターにいくつかの名前付きパラメーターを持たせて、エラーが発生しにくい、または何が起こっているのかを理解しやすくするために、ここで簡単に追加できます。

そのため、checkboxStateとtoggleCheckboxStateがあります。大丈夫。

これで、ListTileの末尾部分にTaskCheckboxが作成され、現在の値を渡します

このチェックボックス状態プロパティを介したisCheckedプロパティ。そして、コールバックであるチェックボックスコールバックも渡しています。そして、このチェックボックスコールバックは、このコールバックへの入力、つまりチェックボックスの最新の状態を渡すときにチェックボックスが変更されるとトリガーされ、それを使用してプロパティの状態を更新します。

つまり、そのプロパティが使用される場所、つまり、テキストウィジェットとtask checkboxウィジェットの両方で、これらのウィジェットの両方が再構築され、更新されるのを見ることになります。

それでは、アプリをそのまま実行してみましょう。

そして、チェックをクリックすると、テキストウィジェットとチェックボックスの両方がステータスを更新することに注意してください。

そして、ここでのチェックボックスコールバックまでです。この場合、実際に名前付き関数を作成していることに注意してください。しかし、通常、コールバックを使用するときは、実際には匿名で使用しているため、名前も戻り値型もありません。

ここでも同様に行うことができます。

したがって、コールバックの残りのすべてを取得し、それを切り取り、ここで置き換えた場合、そのすべてを削除することができます。

そして今、私はここで私のTaskCheckboxに送信され、onChangedプロパティ内で使用されるコールバックを持っています。だから、私たちは今、それが機能することをかなり確信しています。

ただし、復習したい場合や、Dartでのコールバックが実際にどのように機能するかについてもう少し詳しく知りたい場合は、次のレッスンでそれについて詳しく説明します。

しかし、これまでのコードの動作にかなり慣れている場合は、次のレッスンをスキップしてTodoeyアプリの構築を続行できます。そのため、その他すべてについては、次のレッスンで説明します。