そのため、以前のレッスンでは、チェックボックスがチェックされていて、チェックオフされたタスクがテキストウィジェットとチェックボックスウィジェットの両方に反映されるかどうかの状態を解除することができました。

だからそれはかなりきちんとしている。

ただし、現時点では[タスクの追加]画面のユーザーインターフェイスはありますが、まだ機能していません。

ここに何かを追加して[追加]をクリックしても、何も起こりません。もちろん、このフラットボタンのonPressedには、実際には1つのコメント以外のコードはありません。

それでは、これを機能させてみましょう。

最初にやらなければならないことは、基本的に3つのハードコーディングされたタスクを生成するだけなので、タスクをより動的にする必要があります。

そして彼らは皆、「これは仕事だ」と言います。 To Doリストアプリにはあまり興味がありません。

それでは、どうすればそれをもう少し動的にすることができますか？

まず、タスクリストをステートフルウィジェットにして、状態を更新できるようにします。

そして、私たちができるようにしたいのは、そのプロパティが変更されたときに状態を更新できるように、追跡および更新できるものを持つことです。

したがって、タスクオブジェクトのリストを作成し、このリストタスクを呼び出すことができます。

そして、それぞれがタスクの名前のプロパティとそのタスクが完了したかどうかを持つタスクのリスト全体に等しくなるように設定されます。

それでは先に進んでlibにモデルフォルダーを作成しましょう。先に進んで新しいパッケージを作成します。これをモデルと呼び、モデル内でtask.dartという新しいDartファイルを作成します。

そして今、この新しいファイルの中に、task（singular）と呼ばれる新しいクラスを作成し、このクラスはすべてのタスクがどのようになるかを記述します。

そのため、タスクの名前になる文字列プロパティがあり、そのタスクが完了したかどうかになるブールプロパティもあります。

そして、Taskオブジェクトが初期化されたら、名前を渡し、プロパティが完了したかどうかを確認する必要がありますが、デフォルトではほとんどすべての新しいタスクは正しく実行されません。

それ以外の場合、なぜ作成するのですか？そこで、等号を追加してデフォルト値を設定し、そのデフォルト値をfalseに設定します。

したがって、すべてのタスクは、isDoneプロパティをfalseに設定して開始されます。

ここまでは順調ですね。

タスククラスにメソッドを指定することもできます。このメソッドは単純にtaskcの処理状態を切り替えるため、toggleDoneと呼びます。

この場合、必要なのはisDoneプロパティの内部を調べて、現在のプロパティの反対側に設定することだけです。

したがって、これがtrueに等しい場合、その感嘆符はtrueの反対に設定され、falseになります。

これは非常に簡単です。そして、タスクリストで使用できるタスクモデルを取得し、日付タイプとして既にタスクのリストを追加しましたが、そのモデルフォルダーとtask.dartファイルをインポートする必要があります。そのTaskオブジェクトを認識できるようにするため。

タスクのリストを取得し、画面が完全に壊れているので、ここでコードを完成させるまで心配する必要はありません。そして、新しいタスクオブジェクトのリストを作成します。

とりあえず空になりますので、あと2回だけ貼り付けます。そして、これらのタスクはそれぞれ名前プロパティを取得します。

それでは、「牛乳を買う」、「卵を買う」、「パンを買う」と言いましょう。

大丈夫。

それで、購入される食品がたくさんあります。そして今、タスクのリストがあります。

しかし、このタスクのリストをタスクタイルのリストに変換するにはどうすればよいでしょうか？

現在、私たちのタスクタイルは実際には非常にインテリジェントなものです。

これらは、すべての機能が組み込まれているため、ステートフルウィジェットです。

しかし、先に進み、それをより簡単にしましょう。

これらのコード行をすべて削除し、ステートレスウィジェットに変更します。そして、今ではステートフルウィジェットがなくなったため、さまざまなプロパティを持つことはできません。そのため、ここで少し警告を受けています。

したがって、これをfinalプロパティに変換する必要があります。また、設定する状態がないため、これ以上状態を設定できません。

そこで、このコールバックを取り出して、ここに貼り付けて、タスクチェックボックスの代わりに、チェックボックスウィジェットを元に戻して使用します。リストタイルの最後の部分に表示されるウィジェット。

理由は、2つのステートレスウィジェットがあるため、リストタイルとチェックボックスを分離することは、テキストウィジェットとチェックボックスの両方が再構築されるため、リストタイルを変更するたびに実際には意味がないためです。

そのため、この場合はそれらをまとめておくことが理にかなっています。

さらに、この例を使用して、同じページで見やすい簡単な方法で状態を上げてコールバックを渡す方法についてすべて学習しました。

これで、このタスクのチェックボックスを削除する準備が整い、沼地の標準のFlutterチェックボックスのみを使用できるようになりましたが、同じアクティブな色と値、およびonChangedを使用しています。

そのため、コールバックとして使用されていたコードをコメントアウトし、今のところ「onChanged」もコメントアウトします。私たちは戻って少し後で対処します。

しかし、チェックボックスの値としてisCheckedプロパティを使用します。また、新しいタスクタイルを作成すると、isCheckedプロパティが初期化されます。

そのため、this.isCheckedを渡します。もう1つ渡したいのは、リストのタイトルです。

「これはタスクだ」といつも言ってほしくありませんよね？

そのため、新しいリストタイルを作成するときに、ここに入力するタスクタイトルを指定できるようにしたいと思います。もちろん、これはtaskTitleという別の最終的なStringプロパティになり、isCheckedのような値を持ちません。プロパティ。

そのため、最初は両方ともnullになりますが、このクラスを作成するときに初期化されます。

そのため、ここにもthis.taskTitleを追加します。

タスクタイルを非常にシンプルなステートレスウィジェットに変換しました。isCheckedプロパティまたはタスクタイトルプロパティのいずれかを更新するたびにリストタイルを再構築するだけで、タスクタイルを構築するときに渡されます。 。

それでは、タスクリストに戻って、ここで必要な引数を追加しましょう。

タスクのタイトルはタスクリストから取得されるため、「タスクを作成します」と、角かっこゼロを使用してリストから最初のアイテムを引き出し、次に名前プロパティをタップしますタスクオブジェクトをタスクタイルのタスクタイトルに割り当てます。追加する必要があるもう1つのプロパティはisCheckedプロパティで、これもタスクの0番目のアイテムになり、これはisDoneプロパティになります。そして、私は単にこれをコピーして3つのタスクタイルすべてに貼り付けますが、もちろん、タスク内の次のアイテムを取得するためにインデックスを更新します。したがって、このタスクタイルには、（タスクリストの最初または0番目のアイテムからの）情報が表示されます。これは最初のアイテムからの情報で、これは2番目のアイテムからの情報です。

これで、アプリを再実行すると、タスクリストからタスク名が引き継がれていることがわかります。

しかし、ご想像のとおり、リストビューに新しいタスクを追加するたびにこのウィジェットのリストに新しいタスクタイルを追加する必要がある場合、それは非常に多くの労力であり、非常に多くの計算です。

そこで、このレッスンでリストビューを作成する別の方法、つまりリストビュービルダーについて説明します。

これでリストビューが既に構築され、現在どのように機能するかがわかりました。

しかし、このリストビュービルダーは、リストビュー内に多数の子を構築する必要がある場合に実際に役立ちます。これは、実際に表示されている子に対してのみ呼び出されるためです。

そのため、リストが非常に大きい場合にリソースを節約できます。

したがって、無限のTo Doリストがある場合、または2,000個のタスクタイルが含まれているとわかっている場合、実際にはリソースを消費して、画面上に表示されるものを構築するだけです。多くの計算を保存し、アプリを作成します。より速く。

それでは、このリストビュービルダーはどのように機能しますか？

このビルダーで唯一必要なプロパティは、インデックス付きウィジェットビルダーです。インデックスウィジェットビルダーは、基本的に、現在のビルドコンテキストと、構築中の現在のアイテムのインデックスという2つの入力を受け取るコールバックです。

それでは、私たちのケースでこれがどのように機能するかをお見せしましょう。ここでは、タスクタイルの繰り返しがたくさんあるので、このコードをすべて使用して、すべてを入力して指を傷つける代わりに、先に進んですべてを削除します。

代わりに、ListView.builderをタップします。そして、コールバックをアイテムビルダープロパティに渡します。

そこで、通常どおり、コールバックを使用してかっこと中かっこを追加し、その中に現在のビルドコンテキストとインデックスの2つの入力を取ります。

コンテキストはビルドコンテキストになります。

そのため、ステートフルまたはステートレスウィジェットであるビルドメソッドにビルドコンテキストが常にあるように、このビルドコンテキストには基本的に、ウィジェットツリー内の親がどこにあるかを子ウィジェットに伝える情報が含まれています。

そのため、タスクリストは、展開されたウィジェット内のコンテナ内にあります。コンテナは、それ自体の列ウィジェット内にあり、それから足場などにあります。そして、それがリストビルダに渡され、子ウィジェットを正しくレンダリングし、ウィジェットツリーのどこにそれをスロットするかを知ることができます。

これで、インデックスはすぐに使用するプロパティになります。

したがって、アイテムビルダーに最初に追加するのは、作成するものを指定することです。

そのため、タスクタイルを返すことになります。もちろん、タスクタイルは名前付きパラメーター、タスクタイトル、およびisCheckedプロパティを取得します。したがって、リストビューをかなり手動で作成するときにそれを確認しました。タスクのタイトルはタスクリストから派生するので、タスクリストにアクセスします。このリストから抜き出すハードコーディングの代わりに、単にインデックスを渡します。このコールバックから取得され、現在作成中のリストビュー内のインデックスと一致します。

したがって、ここでこれを作成している場合、渡されるインデックスはゼロに等しくなります。これを作成する場合は1などになります。

これは実際にはタスク型であり、タスクのタイトルが必要な文字列と一致しないため、これでは十分ではありません。

そのため、nameプロパティを利用する必要があります。これは以前とほぼ同じですが、この場合、ビルダーからのインデックス値を使用しているため、動的になりました。

これで、タスクリスト内のすべてのアイテムのタスクタイルを動的に作成するリストビュービルダーができました。

しかし、もちろん、何個のアイテムを構築するかを知ることはできません。

そのため、このリストビルダーに加えて、ビルダーのアイテム数である別のプロパティを追加することもできます。したがって、これらのタスクタイルを実際にいくつ構築したいかを伝えることができます。

そしてもちろん、所有するタスクの数に応じた数のタスクタイルを構築したいと考えています。したがって、単に動的な値を取得するためにtasks.lengthを追加するだけなので、今は3に等しくなります。

そのため、リストビュービルダーのアイテム数も3になり、関連するタスクを持たない何かを構築しようとしていたエラーを取り除きます。

これでまったく同じことができましたが、この場合、リストビューは完全に動的になり、タスクリストの状態を更新するとすぐにリストビュービルダーが再構築されます。しかし、ユーザーが実際に見ることができる部分、つまり画面上の部分だけを再構築します。

1000個のタスクがある場合、リソースを使い切って画面上に1000個のアイテムを作成するのではなく、ユーザーが見るだけの数だけを作成するので、はるかに効率的です。これで、ようやくタスクタイルに戻り、チェックリストを修正する方法を見つける準備ができました。動的リストビュービルダーを機能させることに焦点を当てるために、その機能を壊したためです。

最初に、チェックボックスが親に変更されたときにこの情報を渡すためにコールバックが必要になります。したがって、この場合、親はタスクタイルの親になります。

ここで、前に作成したコールバックを再び使用でき、ここでonChangedメソッド内で使用できます。

そのため、このメソッドを使用して、タスクリストに戻します。

ここで名前付きメソッドを作成するか、もっと簡単に作成することができます。そして、より頻繁に表示されるのは、匿名関数として単純に追加できることです。そして、タスクタイルを作成するときに、そのコールバックを渡します。

そこで、ここではまだ存在しない新しいプロパティを作成し、それをCallcheckboxCallbackと呼び、この中に以前から持っているコールバックを貼り付けます。

そのため、チェックボックスがブール型の現在のチェックボックス状態で渡されるタスクタイル内でトリガーされるため、これはオプションです。そして、私たちはそれを持っていてもいなくてもかまいません。

そして、チェックボックスのいずれかをクリックすると、このコールバックがタスクタイル内でトリガーされ、構築中の現在のインデックスでタスクを更新する最新の値が渡されます。完了状態を切り替えます。

そのため、タスクモデルの内部では、isoneDoneプロパティを以前とは逆に変更するだけのtoggleDoneというこのメソッドがありました。

そのため、これを設定状態で呼び出し、現在のタスクタイルの再構築をトリガーし、更新されたユーザーインターフェイス、つまり取り消し線のテキストとチェックボックス表示を取得することを望みます。ただし、現在のところ、チェックボックスコールバックプロパティはまだ存在していません。

それをコピーして、タスクタイル内に作成しましょう。

したがって、checkboxCallbackと呼ばれる最終関数を作成し、タスクタイルを初期化するときにも初期化を行います。そのため、これをいくつかの方法で行うことができます。私のチェックボックスが変更されたときに、このコールバックに渡される新しい値が追加されると言うこともできます。その場合、checkboxCallbackを呼び出して新しい値を渡すことができます。

しかしもちろん、これは入力を受け取り、出力を返さないコールバックがあるため、このように冗長です。

また、ここには、入力を受け取り、戻り値を持たない関数をトリガーしようとするonChangedプロパティがあります。

そのため、実際にはonChangedのプロパティ内でcheckboxCallbackを使用するだけです。

そして今、保存を押してアイテムをチェックアウトすると、この機能が以前とまったく同じように機能していることがわかりますが、唯一の変更は、動的リストビュービルダーがあり、これをビルドする準備ができていることです必要に応じて、リストビューの多くのアイテム。

次のレッスンでは、タスクタイルを動的にロードする機能を備えたため、タスクリストへのアイテムの追加を開始し、すべてのユーザーインターフェイス要素の状態を更新できるようにするため、課題があります。アプリを更新して、新しいアイテムを前のアイテムの下に追加できるようにします。したがって、すでに次のレッスンに進んで、チャレンジを完了してください。