プロバイダーパッケージのすべてについて学習したので、保存したタスクを読み取ることができるように、appで実際に実行します。

新しいタスクを追加したり、タスクをチェックオフしたり、ユーザーが希望するときにタスクを削除したりできます。

プロバイダーを実践するこの最初の部分では、タスクを取得し、タスク画面とタスクリストの両方でそれを使用できるようにするという最初の目標に取り組みます。

基本的に、タスクプロバイダーの状態を高く保ち、ツリー全体で使用できるようにし、そのデータをTaskScreenとTasksListの両方に提供します。タスクデータの変更をリッスンします。タスク画面では、必要な理由は、12個のタスクと表示されている小さな部分があるためです。

その数は動的であり、現在To Doリスト内にあるすべてのタスクを反映する必要があります。

そして、タスクリストについては、それはかなり自明です。

このタスクのリストを実際にレンダリングできるように、そこで利用可能なタスクデータが必要です。

プロバイダーを実装する今後のレッスンでは、最初に必要な動作を正確に説明します。次に、チャレンジとしてそれを完了するか、単に一緒にフォローするかを選択できます。

しかし、自分で実際に試してみることをお勧めします。動けなくなって戻ってきたとしても、単に従うだけでなく多くのことを学ぶことができます。

そのため、この場合、変更通知プロバイダーを使用して、ウィジェットツリーのまさにルートにあるアプリの最上部でタスクデータを提供する必要があります。

そして、タスク画面内でそれをリッスンしてその番号を更新し、またタスクリスト内でリッスンしてタスクタイルのリストを更新できます。

そのため、既に作業を終えたら、現在のTodoeyアプリプロジェクトを開いて、この目標を達成してください。

だからあなたはそれを試してみて、あなたが解決策をチェックするためにここにいることを願っています

同じことをする方法はたくさんあることを覚えておいてください。あらゆる種類の異なる方法を使用して、まったく同じ動作を実現できますが、私のソリューションはその1つの方法にすぎません。

したがって、最初に行う必要があるのは、もちろんpubspec.yamlに移動し、プロバイダーを依存関係として追加することです。

ここでそれを行い、プロバイダー関連のすべてのファイルを取得して、使用できるようにします。

最初にやるのは、モデルフォルダ内に新しいDartファイルを作成し、このtask\_dataを呼び出すことです。

したがって、これはツリー全体に提供されるオブジェクトになり、タスクのデータにタップインできるようになります。

このクラスは単にTaskDataと呼ばれ、再び変更通知クラスを拡張します。

現時点でエラーが発生する理由は、foundation.dartパッケージをインポートできるようにする必要があるため、またはmaterial.dartパッケージをインポートできるためです。

ただし、material.dartパッケージには実際にFoundation.dartパッケージが含まれています。したがって、軽量に保ちたい場合は、単にFoundation.dartを使用するだけです。

そして今、foundation.dart内で変更通知機能を見つけ、それを使用してタスクデータクラスを拡張し、このクラスを、他のオブジェクトが変更をリッスンして通知するときに更新できるものに変換できるようにします。 。

最初にやることは、他のモデルクラスをインポートすることです。これはtask.dartであり、簡単なリマインダーは基本的にタスクオブジェクトの構造です。そして今、tasks screen.dartにアクセスすると、現在のタスクデータの場所を見つけることができます。 my TasksScreenState内のタスクのこのリスト内にあります。

そのため、これをすぐに解除するので、ここから切り取り、タスクデータオブジェクトに移動します。

ここに貼り付けられます。

まったく同じ方法で作成されます。タスクには名前があり、デフォルトでは「未完了」に設定されています。これで、アプリ全体でこのタスクプロパティを提供する準備が整いました。そして、ウィジェットツリーの最上部でそれを行います。

そのため、MyAppクラス内でそれを行い、この変更通知プロバイダークラスでmaterial app内にあるすべてのものをラップします。そしてもちろん、それを使用できるようにするには、provider.dartパッケージをインポートする必要があります。

そのため、プロバイダーは、後で分岐するウィジェットの残りの部分から見えるほど十分に高いツリー内の場所にいる必要があることを覚えておいてください。

したがって、この場合、それはMyAppクラスの最上部に配置することを意味します。

ここで、データがタスクタイル内でのみ必要であり、他に何も必要ない場合、1レベル上で親に移動し、その親であるこのタスクリストをラップすることができます。プロバイダーウィジェットを使用して、そのすべての子に提供できるようにします。

しかし、これを間違え、ツリー内でこれを低くしすぎると、子供たちがアクセスできなくなる可能性があり、アプリを実行して、正しい位置に配置されていないことを伝えることを試みるとエラーが発生しますウィジェットツリー内。

だからこれがどこに行くかに注意してください。

私たちの場合、素材アプリの上にある必要があります。

必要な変更通知プロバイダーのプロバイダーと呼ばれるパラメーターがあるため、ここで少し警告が表示されます。

それで前に見ました。ビルダーは現在のコンテキスト、つまりウィジェットツリー内の現在の位置を取得し、ツリー内のすべての子に提供する必要があるオブジェクトを返します。そして、そのオブジェクトはもちろん新しく作成されたタスクデータオブジェクトにあります。

それでは、先に進んでタスクデータオブジェクトを提供し、modelsフォルダーから来るこのタスクデータオブジェクトについても伝えましょう。これはtask\_data.dartと呼ばれます。

そのため、ウィジェットツリーの最上部に変更通知プロバイダーがあり、そこにあるデータをリッスンするツリーの下のすべての子にタスクデータオブジェクトを提供します。ここから実際にデータを読み取ることができます。つまり、タスクリストです。

そのため、最初に行う必要があるのは、実際にはタスク画面です。ここでは、リストに残っているタスクの数をユーザーに示すすべてのtasks.lengthがありますが、これはタスクリストに基づいている必要があります。

ここにいくつあるかを数え、ここで補間された文字列として提供する必要があります。

また、タスク画面の状態からタスクを削除したため、現在エラーが発生しています。

先に進み、tasks.lengthを削除し、以前に学んだことを使用してプロバイダーにアクセスし、.ofメソッドを呼び出します。

ここで、Android Studioにコードの入力を手伝ってもらいます。

しかし最初に、provider.dartをインポートする必要があることを意味します。

そして今、私はプロバイダーの.ofメソッドをタップし、もちろんメソッドの日付タイプを常に指定することができます。そして、それは再びインポートする必要がある私のタスクデータオブジェクトになります。

そこで、モデルを活用し、task\_data.dartを活用します。

最終的には、task.dartの必要性を取り除き、関連するすべてのものをタスクデータ内で管理します。

しかし、今は先に進んで、このタスクデータオブジェクトと同等になるこのオブジェクトをタップして、タスクプロパティを取得します。 .tasksを作成して、現在3に等しいオブジェクトのカウントまたは長さを取得します。

それでは、保存して、Dart分析エラーが発生している部分を削除してみましょう。

したがって、最初の部分は、現在存在しない現在のタスクリストにタスクを追加するこの部分です。もちろん、未定義です。

したがって、実際に先に進み、セットの状態をコメントアウトします。次のレッスンでは、タスクデータプロバイダーに新しいタスクを追加する方法に取り組み、この部分を理解します。

したがって、他の部分はもちろん、ここからタスクリストにタスクが渡されます。

これはもちろん、タスクデータから読み取る必要がある2番目の場所です。

.taskプロパティですが、それを渡してツリーをドリルスルーする代わりに、単にタスクリストに移動して、代わりにここでそのタスクデータオブジェクトへの参照を取得します。

したがって、実際にもう何も渡す必要はないので、これらの2つのことは必要なく、task.dartへのこの参照も必要ありません。

ただし、代わりにprovider.dartをインポートし、モデルのフォルダーであるtask\_data.dartもインポートします。そして、タスクリスト内でそれを使用して、リストビューを作成します。

ここで、。タスクリストステートフルウィジェットに渡されるプロパティを利用するwidget.tasksを使用する代わりに、単にprovider.ofメソッドを利用するだけです。そのため、widget.tasks。[index] .nameの代わりに、widget.tasksをprovider.ofに置き換え、タスクデータであるデータ型を指定します。そして、provider.of.tasks [index] .nameになります。これをコピーして、これらすべての場所でウィジェットを置き換えることができます。

そして今のところ、私は実際に設定状態をコメントアウトして、チェックボックスがオンになったときに何が起こるかを処理します。

これについては後のレッスンで扱います。

しかし、現時点で必要なのは、プロバイダーパッケージを使用して提供されるタスクデータ内のタスク内のデータを読み取ることができるようにすることだけです。

ステートフルウィジェットに由来するウィジェットを使用していないことに注目してください。

これが、削除したプロパティにアクセスする方法です。

したがって、実際には、これをステートフルにする必要はありません。

実際にそれをよりシンプルなステートレスウィジェットに変えることができますが、コンストラクターを介してタスク画面から渡すことはないので、「プロバイダー」に追いついて必要なデータを取得することができます。

タスク画面内に状態も必要ないので、これらの行をすべて削除してステートレスウィジェットに変更することで、ステートレスウィジェットにも変換できます。未使用のimortをここで削除できます。同様に。

必要なときに構築され、更新されると破棄され、新しいウィジェットに置き換えられるメモリレスのステートレスウィジェットがあるため、実際にはアプリは非常にシンプルになりました。

そのため、タスクデータのプロバイダーをタップして、ここでそのリストを取得して、タスクのタイトルを設定します。また、タスク画面内でそれを使用して、残っているタスクの数を表示しています。

そして、これは単にタスクを読むだけです。

先に進み、保存をクリックしてアプリをゼロから実行しましょう。そして、うまくいけば、すべてが順調に進み、すべてのコードが適切な場所に配置されることを期待できます。以前とまったく同じユーザーインターフェイスが必要です。しかし、今回は、このデータはもはや渡されず、ツリーを介してドリルダウンされず、代わりにプロバイダーから取得されます。

それで、あなたが自分でそれに挑戦しようとしたとき、あなたはこの点に到達することができましたか？

変更通知を拡張するタスクデータクラスを作成し、そのタスクデータオブジェクトを提供する変更通知プロバイダー内にマテリアルアプリをラップし、最後にタスクデータプロバイダーを使用してタップして使用するなど、いくつかのことを行いましたタスク画面とタスクリストの両方に表示されます。これは今でも機能しますが、実際にはもっと簡単にすることができます。また、provider.dartから提供されるコンシューマウィジェットと呼ばれるものを使用してこれを行うこともできます。ここでは、provider.of task data.tasksを3つの場所で使用していることに注目してください。

少し繰り返しますよね？

したがって、provider.ofを3か所で呼び出す代わりに、このタスクプロパティがコンシューマウィジェットと呼ばれるものの内部で変更されたときに更新する必要があるすべてのダウンストリームウィジェットをラップするだけです。

タスクタイル内でprovider.ofを使用していることに注意してください。ただし、アイテムカウント内でも使用しています。

したがって、それより1レベル上に移動すると、それがリストビューウィジェットになります。

それでは、リストビューウィジェットを、コンシューマウィジェットとなる新しいウィジェットにラップしてみましょう。

そして、これはもちろんプロバイダーパッケージに由来します。

そして、それを使用するために、リッスンしている特定のデータが変更されたときに更新する必要があるダウンストリームウィジェットをラップし、タスクデータになるデータタイプを指定します。

それが私たちが聞いていることです。

そして、それは単にビルダーを必要とし、ビルダーは現在のコンテキストを取り、

そのため、ツリー内のどこにいるかに加えて、さらに現在のデータを提供し、そのオブジェクトに名前を付けることができます。

これがタスクデータオブジェクトになります。

したがって、単にtaskDataと呼びます。これは、タスクデータコンテキストのprovider.ofを呼び出すのと同等または基本的に同じです。

そして最後に、子と呼ばれるプロパティも取ります。

これで、ビルダーが構築され、このデータを使用して構築する必要のあるウィジェットを簡単に返すことができます。子プロパティを持つ代わりに、このリストビュービルダー全体をリストビュービルダーの最後のコメントまで引き下げます。それを切り取り、それを必ずコンシューマー内に戻します。もちろん、このウィジェットを返すので、コンマの代わりにセミコロンをここに追加する必要があります。これで、すべてのエラーがなくなりました。provider.ofblah blah blahを使用する代わりに、すべてをtaskDataに置き換えることができます。

次に、provider.ofがあるすべての場所でtaskDataを置き換えます。今度は、このコンシューマウィジェットがタスクデータの変更をリッスンするウィジェットになります。

そのため、タスクデータの状態が更新されると、このリストビュー全体が再構築され、変更されたすべてのタスクタイトルが更新されます。

これで、コードははるかに簡潔になり、さらに先に進むことができます。なぜなら、ここでtaskData.tasks.lengthを使用しているので、ここでタスクリストの長さを取得しているからです。ここでもそうですか？

したがって、通常、同じことを何度も繰り返している場合は、おそらく何かをセットアップして、次回は少し簡単に、もう少し便利にするとよいでしょう。

したがって、タスクデータにアクセスすると、ゲッターを持つ新しい整数を実際に作成できます。

そのため、get toキーワードを使用してこれをゲッターと定義し、ゲッターは値を返すものになります。

つまり、誰かがこのプロパティの名前、このtaskCountをタップすると、計算する値を返すことになります。そして、タスクリストに基づいてこの値を返します。

つまり、tasks.lengthと言います。そして、このタスクカウントプロパティを利用しようとするたびに、このリストであるタスクを見てその長さを取得することで、その値を計算します。そのリスト。

したがって、タスク画面に移動して、task.tasks.lengthのプロバイダーを呼び出す代わりに、単にタスクカウントに単純化することができます。もちろん、タスクリスト内でも同様に行うことができます。 taskCount。

そのため、多くのprovider.ofを節約できるコンシューマウィジェットを使用して、コードを劇的にリファクタリングしました。

また、タスクリストの長さを取得するプロセスを簡素化するために、タスクデータ内にゲッターを作成することができました。

もちろん、この課題を完了するためにこれをすべて行う必要はありませんでした。

私が望んだのは、プロバイダーパッケージを使用して、変更通知機能を使用および拡張する新しいオブジェクトを使用することで、提供するデータを取得できること、および変更通知プロバイダー内でツリー全体をラップできることを確認することだけでした。 providr.ofを使用してタスク画面内でこのタスクデータオブジェクトを提供できます。または、これまで見てきたように、コンシューマウィジェットも使用できます。

これで、プロバイダーの読み取りと公開が完了しました。

しかし、次のレッスンでは、変更通知機能を使用して、プロバイダーに新しいタスクを追加する方法を見ていきます。

それで、それ以上については、次のレッスンで説明します。