このリストで考えてみると、すでにセット状態に遭遇しているので、それを使用してローカル状態を非常によく制御および管理する方法を知っています。もちろん、ステートフルウィジェットと組み合わせて使用​​する必要があります。ただし、ステートフルウィジェットはステートレスウィジェットよりもはるかに複雑であるため、1つのステートフルウィジェットには実際にメモリが必要です。

ステートレスウィジェットは操作がはるかに簡単であり、作成、破棄、および再構築することができるのに対し、変更できるように状態を覚えておく必要があります。

また、コールバックを組み合わせて使用​​し、ウィジェットツリーで状態を解除することで、以前に状態を解除することも確認しました。状態を管理することができました。しかし、もちろん、アプリがより複雑になり、ツリー全体を横断するコールバックが必要な場合、または状態を複数レベル下に渡す必要がある場合は、いくつかの問題が発生する可能性があります。

そのため、現在、状態を管理する最も柔軟な方法は、プロバイダーパッケージを使用することです。

これは現在、Google Flutterチームからの推奨事項です。

しかし、私が言おうとしているのは、これが公式に推奨されている方法であるにもかかわらず、Flutterアプリで状態を管理する方法であることに留意してください。

また、多くの場合、別の方法が必要な場合や、状態を管理するこれらの異なる方法の組み合わせが必要な場合があります。

それは間違っていません。

それは本当にあなたのユースケース、つまりあなたの要求、あなたのコードをどのように整理したいか、そしてFlutterのフレームワークの成熟度に特有のスタイルの感覚に依存します。

しかし、それは本当に素晴らしいツールだからです。

これは、実際にコミュニティによって作成されたパッケージです。

グーグルはProvidと呼ばれる似たようなパッケージを持っていましたが、これは何か似たようなことをしましたが、彼が来たので、彼はレミと彼は実際により良いと思われるバージョンを構築することにしましたそのため、「Google Flutterチーム」への全面的な功績は、このパッケージの可能性を実感し、実際に全力を尽くしました。

そして今では公式に推奨されているだけでなく、レミ自身とGoogle Flutterチームの両方からもサポートされています。

そのため、その潜在能力を最大限に引き出すために必要なすべてのリソースを備えています。

ここで、プロバイダが非常に有用である理由を完全に理解するために、最初に実際に解決しようとしている問題を検討する必要があります。

最初に私たちのコードから、あなたはすでに少し毛むくじゃらになっているものがいくつかあることにすでに気付いているかもしれません。

ツリーにデータを渡すためにこれらのコールバックがすべてあり、set dateを使用して子ウィジェットのビルドメソッドを呼び出す必要があるステートフルウィジェットの束があり、多くのものが常に再構築されています。そして、おそらくこれは最も効率的な方法ではありません。

しかし、私たちのアプリは非常に大きくなり、状態を管理しているときに何が起こっているのかを一目で見るのは非常に難しいため、問題を少しよく説明できるように本当にシンプルなアプリを作成しました。

したがって、この場合、MyAppのトップレベルクラスがあり、2つの子があり、そのうちの1つはレベル1と呼ばれます。次に、レベル1はレベル2とレベル2を作成し、レベル3を作成します。

そのため、そのコードは次のようになります。それは非常に簡単です。重要な部分は、レベル1クラスを作成し、次にレベル2およびレベル3クラスをこの順序で作成するMyAppクラスがあることです。

レベル1を作成し、レベル1がレベル2、レベル2がレベルLevel 3を作成するマテリアルアプリです。

基本的にまさにこの構造です。

ここに問題があります。

ここだけでなくここでも使用する必要があるプロパティがあったとしましょう。そのためには、状態を解除してアプリ内で動作させる必要があります。

これで非常にうまくいきましたが、そのプロパティまたはそのプロパティの状態をレベル3に渡すと、そこに到達するためにレベル1とレベル2とレベル3を通過する必要があります。つまり、そのデータはこれらの2つのクラス内で更新されますが、実際には必要ありません。

したがって、下位レベルに到達するには、基本的にツリーをドリルダウンする必要があります。そして、リアクティブプログラミングでは、これはプロップドリルと呼ばれることもあります。そのため、プロパティを継承するためにツリーをドリルします。これを実際のコードで見てみましょう。

単にデータと呼ばれる文字列があり、それが「トップシークレットデータ」だとしましょう。

そして、これは両方のAppBar内で使用する必要があります。したがって、タイトル内でテキストウィジェットとして使用され、そのデータが含まれます。ただし、ここではレベル3でも使用する必要があるため、このレベル3のテキストウィジェットにもデータが必要です。しかし、レベル3に移行するには、すべてのレベルをドリルスルーする必要があります。

したがって、レベル1の内部では、データを呼び出すだけのプロパティが必要になり、レベル1を初期化するときに、このデータプロパティを渡すことで初期化することになります。そして、コンストラクタを介してレベル1に渡し、データがレベル1内にあるようにします。

しかし、レベル3に下げるには、レベル2に渡し続ける必要がありますか？

それをレベル2に渡して、dataという名前の最終的なStringプロパティを作成し、レベル2が初期化されたら、このプロパティをthis.dataに渡します。

そして最後に、レベル3に渡すだけです。

したがって、最終的な文字列データを作成し、レベル3を初期化するときに、必要なデータの値を渡します。

そして、最終的に、レベル3で必要なデータを取得し、テキストウィジェット内に実際に配置できるようにしました。

それで、保存をクリックしてアプリを実行すると、正しく動作することがわかりますか？

私のトップレベルのウィジェットであるMyApp内にある私のデータは、このデータプロパティを私のAppBarと、テキストウィジェットを持つこのレベル3ウィジェットの両方に渡すことができます。しかし、それを行うために、このデータをまったく関係のない、本当に関係のないウィジェットの束全体に渡さなければなりませんでした。レベル2では、レベル1のデータは必要ないことがわかります。そのデータは必要ありません。レベル3に下げるために、このデータをそのまま渡します。

怠け者のプログラマーにとっては、これは見るのがかなり苦痛であり、また複雑さを大幅に増加させます。

ここにタイプミスがあったり、レベル1内に他のデータがあると言ったり、「渡されるデータとローカルバージョンのデータとが混同される」と言ってください。本当にきれいではありません。

では、代わりに何ができますか？

Providerパッケージがある場合は、状態を最上部に移動できますが、状態にサブスクライブするだけのウィジェットを作成できます。そのため、レベル3のウィジェットに手を上げて、状態の変化をリッスンするように言ってもよいでしょう。そして、それが変更されると、私は通知されます-レベル3に到達するためにレベル1とレベル2を通過する代わりに、レベルを再構築する代わりに、この更新された状態を実際に処理するレベル3ウィジェットにデータを渡すだけです1、レベル2およびレベル3。

そのツリーを再構築して状態を渡し、実際にレベル3に到達させる代わりに、単にレベル3に更新をリッスンさせるだけです。

では、プロバイダーを使用してこれらすべてのクールな機能を取得するにはどうすればよいでしょうか？

まず、アプリにプロバイダーパッケージを含める必要があります。

それでは、インストールに戻り、プロバイダーを依存関係として追加しましょう。

pubspec.yamlを開き、Cupertinoアイコンのすぐ下にすべての依存関係があります。プロバイダーパッケージを追加し、パッケージを呼び出してそのパッケージからコードを取得します。

そして今、provider.dartファイルをインポートできるはずです。

これで、コードにプロバイダーができたので、それを使用できます。

最初にやらなければならないのは、データを公開することです。つまり、提供するデータは何かということです。それを行うために想像できるとおりで、私たちはフラッターランドにいるので、ウィジェットを使用して実現します。そして、AppBarのデータとレベル3の両方のデータが必要なため、必要に応じてそのプロバイダーを上位に配置します。

それでは、マテリアルアプリ全体にデータが提供されるように、最上部に配置します。

そこで、ここで新しいウィジェットでラップし、そのウィジェットをプロバイダーウィジェットにします。

プロバイダーウィジェットでは、このプロバイダーが提供するデータの種類のデータ型も指定する必要があります。このデータ型は文字列になります。データは本当に単純な文字列だからです。

プロバイダーウィジェットを作成し、後続のすべてのレベルが提供するデータをリッスンできるようにする子があるため、プロバイダービルダーであるもう1つのプロパティを追加する必要があります。

ビルダーは、現在のビルドコンテキストを使用して、ウィジェットツリー内のどこにあるかを認識し、すべての子が利用できる必要があるデータを返す必要があります。それがここのデータになります。

ツリーの最上位でデータが提供されるようになったので、プロバイダーに必要な場所にデータを提供するように依頼できます。

それでは、まずこのコードを単純化しましょう。

単純な1行のリターンがあるので、この中括弧などとreturnキーワードは必要ありません。

太い矢印を使用するだけで、まったく同じことができます。

そのため、マテリアルアプリの上にアプリの最上位レベルでデータを提供しているので、必要に応じてデータにアクセスして使用できます。

最初に必要なのはテキスト内です。

そのため、このデータを実際に必要としないレベル1レベル2ウィジェットにデータを渡す代わりに、実際にこのコードのすべてをこことこことここですべて削除することはできません。

代わりに、レベル3内で使用するだけです。

そして、そのデータにアクセスするために必要なことは、provider.ofだけです。また、「of」メソッドは、取得しようとしているデータの種類を指定することを期待しています。

これは、ツリー内のすべての子にここで提供する文字列になります。

そのため、保存ボタンを押してアプリを確認すると、トップシークレットデータがレベル3のステートレスウィジェットに渡されていることがわかります。

そして、それはここから来ます。ご覧のとおり、このデータを変更することに決めたとしましょう。私は知りません。極秘コードを言ってみましょう。

そして今、アプリを再実行すると、レベル1とレベル2をドリルスルーする必要なしに、データが実際にレベル3に渡されていることがわかります。これですべてうまくいきました。

最上位レベルでデータを提供できたので、ツリー内の中間部分を経由せずに、ここからずっとデータを読み取ることができました。

そのため、ここでそのプロパティの状態が変更されると、レベル3はそれらの変更をリッスンしているため、レベル1またはレベル2を再構築する必要なく、すぐにそれを受け取り、それ自体を再構築できます。

さて、これをもう少し複雑にしましょう。現時点では、プロバイダーで行っていることはすべて提供し、読んでいるからです。ツリー内のどこかから状態を更新する必要がある場合はどうなりますか？

単純なアプリを更新して、レベル1、2、および3を保持しますが、ここに別のクラスのMyTextというテキストウィジェットがあり、レベル2の子であるMyTextFieldウィジェットもあります。

それで、アイデアは、ここで下の方にある更新をMyTextに到達させる方法です。MyTextは、ツリーの上位にあり、電源プロバイダーを使用してレベル3にあります。

そのため、ここにはMyTextクラスとMyTextFieldである別のクラスがあり、それらをツリーに挿入します。

したがって、私のテキストはAppBarのテキストウィジェットとして使用されます。

それをMyTextに変更すると、MyTextFieldは私のレベル2ウィジェットの子になります。

この空のコンテナの代わりに、ここにMyTextFieldを追加します。

したがって、基本的にウィジェットツリーのこの正確な構造が得られ、介在するウィジェットを再構築することなく、このデータをここだけでなくここまで取得できる必要があります。

それでは、どうすればいいのでしょうか？

さて、ChangeNotifierと呼ばれるクラスを使用する必要があります。

そこで、ここでデータクラスとなる別のクラスを作成し、このデータクラスでChangeNotifierクラスを拡張します。

現在、このChangeNotifierクラスは実際にはFlutter自体から来ています。

これはプロバイダーに付属しているものではありません。

これは既に存在し、この特定のクラスのリスナーにクラスの更新を通知するために存在します。

それでは、データをデータクラスに移動し、最終的なものにするのではなく、適切に変更できるようにする必要がありますか。

データを更新する必要があります。

それでは、いくつかのダミーデータから始めましょう。「いくつかのデータ」と呼びましょう。そして、以前から持っていた単純な文字列だけでなく、このデータオブジェクトの提供を始めましょう。

そのため、現在存在しないデータを返す代わりに、データクラスのオブジェクトを返します。

このオブジェクトは、文字列データ型を提供するだけの通常のプロバイダーを使用しなくなり、代わりにChangeNotifierプロバイダーになり、データデータ型を提供します。

したがって、ここでデータにアクセスするには、provider.ofを使用し、文字列データ型を使用する代わりに、提供するデータ型を実際に提供する必要があります。そして、このオブジェクト全体、つまりクラスから作成されるこのデータオブジェクトを提供する代わりに、データオブジェクトのデータプロパティを提供します。それは文字列だからです。

そして、その文字列はテキストウィジェットに挿入されて表示されます。

そのため、これをコピーしてMyTextウィジェットにも配置し、レベル3のテキストウィジェットだけでなくアプリケーションバーにもレンダリングできるようにします。

最初にアプリを実行して、アプリバーのMyTextウィジェットとレベル3のテキストウィジェットの両方でデータを実際に提供し、読み取ることができることを確認します。

だからそれはかなりうまく機能しています。

そのため、データを更新するという次の目標を達成する必要があり、MyTextField内で更新します。

そのため、MyTextFieldには、新しい値またはテキストフィールドに入力される新しいテキストにアクセスするコールバックを受け取るonChangedというプロパティがあることがわかっています。

そして、この文字列を更新するには、このテキストをデータクラスに移動できる必要があります。

それでは、データクラス内にメソッドを作成しましょう。何も返さないようにvoidメソッドを作成しましょう。

そして、文字列が変更されます。

それがchangeStringメソッドです。そして、データの代わりに使用する新しい文字列を渡し、新しい文字列に等しくなるようにデータを設定します。そして最も重要なのは、ChangeNotifierを継承しているため、利用可能なメソッドを使用することです。

そして、そのメソッドはnotifyListenersメソッドです。

そのため、データを新しい値に更新したらすぐに、通知リスナーを呼び出して、プロバイダーの状態を通知しているため、プロバイダーをリッスンしているすべてのユーザーがウィジェットを再構築できるようにします。

したがって、onChanged内でprovider.ofを利用し、関心のあるデータのタイプを指定します。これはデータクラスです。

それでは、changeStringというメソッドを利用して、このデータプロパティに挿入するnewStringとしてnewTextを渡します。

したがって、アプリを再実行してここに入り、入力を開始すると、このデータのすべてのリスナー、つまりレベル3のmyTextウィジェットとテキストウィジェットが再構築されていることがわかります。これで、テキストフィールドのテキストを変更するたびに状態が更新されます。

それでは、もう一度コードを確認しましょう。

そこで、このクラス内のデータの一部がデータの一部、つまりデータというプロパティを追跡しようとしているリスナーに通知できるようにするChangeNotifierと呼ばれるものを拡張するDataというこのクラスを作成しました。そして、そのプロパティは、AppBarのこのテキストウィジェット内とレベル3内のこのテキストウィジェットの両方で読み取られています。

したがって、これら2つのテキストウィジェットの外観は、このデータプロパティの状態に関連付けられています。

そして、そのデータプロパティが更新されると、このテキストウィジェットはその変更を反映するために再構築されます。

ここでデータプロパティがここから来て、change Stringメソッド内でそれを更新し、新しい文字列を渡してデータプロパティ内の古い文字列を置き換えます。そして、決定的に重要なのは、notifyListenersを呼び出して、そのデータプロパティをリッスンしているすべてのユーザーが通知を受け、自分自身を再構築して最新の状態を更新できるようにすることです。

これで、このnotifyListenersを呼び出す必要があるため、基本的にこのデータオブジェクトであることがわかっているデータのプロバイダーを単にタップして、単純にそのデータプロパティをタップして、新しいテキストに等しく設定することはできません。

この場合、外部からこの文字列を更新するだけであるため、実際にはnotifyListenersを呼び出していないためです。

このプロパティの値を更新しているだけです。したがって、実際にnotifyListenersを呼び出すように、メソッドを使用して文字列を変更する必要があります。

覚えておくべきもう1つの重要なことは、このprovider.ofメソッドには実際にはlistenというオプションのパラメーターがあることです。

AppBar内のテキストウィジェットであるMyTextが、MyTextFieldのすべての更新で実際に更新されないようにしたいとしましょう。

格納されているデータの元の値を取得したいが、更新をリッスンしたくない、このデータが変更されるたびに再構築したくないとしましょう。

それでは、listenプロパティを追加するだけで、falseに設定できます。

現在、このプロパティはデフォルトでtrueに設定されているため、provider.ofを使用するたびに、リスニングしているものの変更に基づいて自身を更新するリスナーを取得します。

しかし、falseをリッスンすると言ったので、アプリを再実行してみましょう。両方のテキストプロパティは、データの値となるデータが初期化されると、ここから来るいくつかのデータで始まることがわかります。

しかし、テキストフィールドに入力を開始すると、ここのテキストウィジェットは変更をリッスンしていないため、更新も再描画もされません。

したがって、これにより、データの更新をリッスンして再描画したいものと、データの初期値のみを必要とし、それが再び変更されないようにするものを非常に細かく制御できます変更。プロバイダーパッケージの動作を確認したので、「まあ、それはどのように機能しますか？」

それを理解するために、ボンネットの中にポップアップして、舞台裏で何が起こっているのかを見てみましょう。状態を管理するこれらのさまざまな方法のうち、継承されたウィジェットを使用する方法があります。継承されたウィジェットは、基本的には、ツリーの介在部分を再構築することなく、ツリーにデータを渡すことができるウィジェットです。

それはちょっとおなじみのように聞こえますか？

これが、プロバイダーの仕組みです。

基本的にプロバイダーパッケージは、継承されたウィジェットの単なるラッパーです。ただし、継承ウィジェットクラスには、それを構築するための特定の要件があり、それを特定の方法で構築して、依然として効率的であり、期待どおりに実行されるようにする必要があります。

したがって、非常に多くの定型コードがあり、それを使用しているときにどのように機能するかを十分に意識する必要があります。

そのため、継承されたウィジェットクラスを見て、非常にクールなGoogle自転車シャツでフィリップのこのウィジェットの説明を聞いて、舞台裏で実際にどのように機能するかについてもう少し学ぶことをお勧めします。本質的に私たちのプロバイダーパッケージはこのクラスのパワーを活用しているだけですが、それを操作しやすくし、安全で効果的な方法でそれを使用することを強制する方法でそれを行っているからです。

そして、それが完了したら、このウィジェットツリーの独自のバージョンを構築することをお勧めします。 したがって、この場合は別のクラスがうまくあります。このように構成された約6つの異なるクラスがあり、プロバイダーパッケージを使用して、最上位レベルでデータを提供する方法、下位でデータを読み取る方法を確認します レベルおよびChangeNotifierプロバイダーを使用してデータを更新します。

それで試してみてください。このレッスンで使用したコードを確認したい場合は、コースリソースでダウンロードすることもできます。

しかし、次のレッスンでは、実際にプロバイダーを実際のTodoeyアプリにプラグインし、より複雑で現実的なシナリオでそれを確認します。

それで、それ以上については、次のレッスンで説明します。