そのため、前回のレッスンでは、ジオロケーターを機能させて、ユーザーのデバイスの現在位置を低い精度レベルで取得しました。

したがって、2〜3ブロックの範囲のようなものです。

今、私たちがしていることは、その1つの方法を使用することだけですが、これが正しく失敗するケースがたくさんあることを認識しなければなりませんか？

たとえば、ユーザーが現在の位置を取得する許可を与えなかった場合、ポップアップが表示されて場所の許可を求められたときに、彼らはノーと言いました。

また、トンネル内の使用者がGPSアドレスを取得するために実際に信号を取得することが物理的に不可能な場合など、他の問題もあります。

したがって、発生する可能性のあるこれらすべてのエラーを処理できるようにするには、Dartが例外を処理する方法について学習する必要があります。

今、私たちはあなたが悪いコードを書くとき、私たちのコードは通常コンパイルさえしないことを知っています。エラーが実行される前にキャッチされ、エラーに下線が引かれ、コードに問題があり実行できないため、アプリの実行が拒否されることがわかります。

たとえば、myStringという文字列を作成し、この文字列に等しく設定したとしましょう。一重引用符で囲まれた数字は15です。

この段階で、文字列に5番を追加しようとした場合、myStringに5を加えて書き込みます。

文字列と数字を使っていくつかの基本的な算術を試してみましょう。

これは整数であり、これは文字列であるため、それができないことを教えてくれます。

異なるタイプを組み合わせる方法がわかりません。

これは、コンパイル時エラーと呼ばれるものです。

これらのエラーは、開発者がアプリを実行することさえできず、間違いなく警告されるため、開発者がキャッチするのは非常に簡単です。

コードの実行中に予期しないことが発生した場合、どうなりますか？たとえば、Dartにはparseと呼ばれるメソッドがあり、文字列を取得してdoubleに変換できます。

たとえば、double.parseと言うことができ、文字列を提供できます。

そこで、myStringをdoubleとして提供します。

そして、このメソッドはその文字列をdoubleに変換します。

したがって、文字列としての15は数値として15になります。

次に、それをdouble型の変数に格納し、myStringAsADoubleと呼びます。

そのメソッドの出力と等しくなるように設定します。

このメソッドの仕事は文字列をdoubleに変換することなので、エラーはもうありません。

この場合、それは非常に簡単です。単一引用符を削除すると、文字列として15は、二重の権利になりますか？

したがって、これを印刷することもできますし、myStringAsADoubleと言うこともできますし、必要に応じて5つ追加することもできます。

この段階では、すべてのデータ型が本来のものであるため、コンパイル時エラーはありません。

そして、runを押すと、ストリング15がダブル15に変わり、5が追加されることがわかります。

そして、それはコンソールに出力されます。

だからそれはかなり良いです。

しかし、悪いコードを書いて、実際には動作しないものを与えた場合はどうなりますか？

したがって、myStringを 'abc'に変更し、その文字列abcをdoubleに渡して5を追加しようとするコードを取得しようとするとします。

さて、この場合はどうなりますか？

コンパイル時にチェックするのは、入力する入力に対して、期待するデータ型と一致するため、コンパイル時にエラーが発生しないことに注意してください。この場合、解析には文字列データが必要ですソースとして入力し、それがまさに私たちが提供しているものです。文字列を提供しています。

今では、私たちが実際にそれをだまして、実際のテキストを与えて数字に変えようとしていること、つまり人間ができない仕事です。 「abc」という言葉を与えて、「それを数字として返してください」と言ったら、私たちのコンピューターはもちろん、何をすべきか本当に分かりません。

ただし、アプリの実行中にこのエラーが今後どのように発生するかに注意してください。

だからこれは結構です。

そしてこれは大丈夫です。

しかし、テキストabcを数字に変換しようとする時点で、ここでつまずいて失敗します。

そのため、これはアプリの実行中または実行中に発生します。

実行を開始してランタイムを開始すると、キャッチされていない例外が表示されることがわかります。

したがって、キャッチされない例外であり、abcは実際にはdoubleではないことがわかります。

これをdoubleや数値に変えることはできません。

ここでこのメソッドは解析されます。それは、あなたが真実であり、良いことであり、常に数値に変換できる文字列を常に与えていることに依存しています。

しかし、コードが大きくなり、コードが大きくなるにつれて、何が渡されているのかが常にわかりません。そして、私たちはこのように正直な間違いを犯すかもしれません。

そのため、アプリをクラッシュさせたくありません。しかし、代わりに、例外をキャッチするようにします。その質問への答えとして、コードの実行中に予期しないことが発生した場合、どうすればよいでしょうか？

その瞬間に、プログラムはエラーをスローし、そのエラーをキャッチして処理するのはプログラマーとしてのあなたの仕事です。そして、それを行うために、try and catchを使用できます。

したがって、失敗する可能性があることがわかっている部分、たとえばこの部分を取り上げて、tryブロック内にラップします。だから、「これをやろう」と言うつもりです。しかし、失敗した場合は、catchブロック内でエラーをキャッチします。

したがって、catchを提供し、次に例外に名前を付けます。そのため、exceptionと呼びます。ほとんどの場合、これは例外またはエラーのeショートとして書かれています。

このcatchブロック内で、実際にエラーを出力できます。そして今、コードを実行すると、キャッチされていない例外ではなく、例外がキャッチされ、コンソールに出力されることがわかります。

この時点で、何人かが質問するかもしれませんが、違いは何ですか？

同じことが起こりましたか？

あなたは例外があり、コンソールを見ると例外が何であるかを知っています、そのtry and catchブロックに追加することのポイントは何ですか？違いは、例外が発生し、キャッチブロック内でキャッチされない場合、コードがクラッシュすることです。

つまり、アプリがクラッシュし、ユーザーに黒い画面が表示されるか、アプリが終了します。

しかし、もしあなたがそれを捕まえたなら、あなたはそれに対応することができ、先に計画することができます。

Flutterアプリで実際に試してみてください。

未処理の例外がどのようにクラッシュにつながるかの例を簡単に示します。

また、例外から回復してクラッシュを回避する方法を示します。

現在、ロード画面のビルドメソッド内にあるすべてのものを削除します。代わりに、結果に基づいて何かを返しますが、その結果はmyMarginという文字列に基づいており、15になりますが、文字列として返されます。そして、bodyパラメータを含むスキャフォールドウィジェットを返します。スキャフォールドのボディには、赤色のコンテナになり、marginプロパティもあります。もちろん、EdgeInsetsを使用して作成されます。

そして、私は4つの側面すべてにマージンを持たせるつもりです。

ここで使用する値は、doubleを想定しているため、doubleになります。

そして、この解析メソッドを使用して、文字列myMarginをdoubleに変換し、コンテナーのマージンとして使用できるようにします。

この段階で、アプリを実行してみましょう。何が起こるか見てみましょう。

アプリが起動すると、この赤いコンテナがあり、このコード行に基づいて、4つのエッジすべてから15ピクセルのマージンが与えられていることがわかります。

それでは、コードを壊しましょう。

このマージンを、doubleに解析できないことがわかっている実際のテキストに変更しましょう。

この段階で、保存をクリックするとすぐに、ホットリロードが発生し、このコード行が実行されてアプリがクラッシュすることがわかります。

画面に何も表示されず、コンソール内に多くのエラーが表示されます。

今、私はこれに備えており、いくつかの段階で、ダブルに渡せないものが入力としてここに入れられる可能性があることを知っていました。

それでは、例外でそれをキャッチできました。

このコード行を移動して、myMargin AsADoubleと呼ばれるdoubleを作成しましょう。

そして、double.parse（myMargin）に等しく設定します

これが問題のコード行であることがわかったため、これは壊れる可能性があります。このケースでは、これを試してみてくださいと言いますが、失敗した場合は別のことをします。

そのため、例外をキャッチし、例外を出力します。また、レンダリング用の代替ウィジェットを提供します。

それをコピーして、ここに貼り付けてみましょう。

また、足場になり、コンテナを持ちます。

この場合、myMarginのデフォルト値は30になります。したがって、この場合、このマージンをdoubleに変えてみてください。

そして、可能であれば、レンダリングするこのコンテナ内でmyMarginasADoubleを使用しますが、これにエラーがあった場合は、キャッチブロック内でそれをキャッチして、例外をコンソールに出力するだけでなく、代替手段も提供します、別のコンテナを提供します。

この段階で保存ボタンを押してアプリをリロードすると、このコードがまだ失敗していて、コンソールに「Invalid double」という例外が表示されることがわかります。それはabcです。画面にレンダリングされる代替現実を取得しています。

コードを見ると、実際には非常に簡単なことをしています。

myMarginasADoubleの解析にエラーがあり、myMarginに実際に値がない場合、単にマージンのデフォルト値を提供するだけです。

これで、本当に便利なDartの別のコンストラクトを使用して、これをさらに簡単にできます。 2つの代替のスキャフォールドを作成する代わりに、try catchブロックをすべて削除して、本当にシンプルに保ちましょう。

そのため、マージンを2倍にしようとし、エラーがあった場合はそれをキャッチします。

そして、ビルドメソッドは単にscaffoldを試みて返すだけです

このmyMarginをdoubleとして使用します。

それをtryブロックの外に移動してみましょう。ここからは、値なしで開始します。

そのため、nullで開始しますが、この値myMarginを解析して、myMarginをmyMarginasADoubleに変換しようとします。

例外が発生した場合、例外を出力しますが、それでもウィジェットを返します。

この場合、myMarginAsADoubleを使用します。しかし、今すぐコードを実行すると、マージンをdoubleとしてdoubleに変換しようとしますが、テキストの一部であるため、これは失敗し、catchブロックがトリガーされます。

この場合、エラーが発生してdoubleになった場合、myMarginAsADoubleをデフォルト値に変更します。30を右にしますか？この場合、赤いコンテナに30のマージンが与えられ、abcを30に正しく変換できないことがわかっているため、ここから使用していることがわかります。

したがって、実際にトリガーされているのはcatchブロックであり、コンソールにも例外が表示されます。

しかし、実際には、特定の変数がnullに等しいかどうかを確認するためにこのチェックを行うはるかに簡単な方法があり、その場合は代替値を提供します。

そして、マージンプロパティに沿ってここでそれを行うことができます。

そのため、ここで値を提供する代わりに、myMarginAsADoubleと言うことができ、2つの疑問符を追加します。

そして、この構文が言うことは、これが値を持っている場合、つまりnullでない場合、それを使用するということです。

しかし、それがnullの場合、代わりに私の代替を使用してください

そのため、代わりに30を使用します。

そして、このコード行は以前とまったく同じように機能します。

現時点でmyMarginAsADoubleがnullに等しいため、ここでは30のマージンが適用されていることがわかります。

また、キャッチブロックで処理できる例外もスローされます。

ここで、このコード行が実際に成功するように、これを解析可能な実際の数値に変更したとしましょう。

その後、コードを実行すると、マージンが15に変わります。この値を優先的に使用しますが、実際にnullに等しい場合、この演算子は、代わりにデフォルト値があることを確認し、空白の画面やマージンのレンダリングの問題が発生しないようにします。そのため、このレッスンでは、tryブロック内で、失敗してエラーをスローする可能性のあるコード行を囲むことができるtry catchブロックを見ました。そして、エラーがスローされた場合、catchブロック内でキャッチします。そして、例外を印刷するか、画面に表示してユーザーに警告するか、catchブロックで代替の現実を提供することで、例外を処理できます。

また、null対応演算子も見ました。この場合、ある時点でnullになる可能性があることがわかっている変数があり、値が含まれていない可能性があります。

したがって、その場合、2つの疑問符を追加して、将来この変数を使用しようとしてnullと等しい場合は、代わりにこのデフォルト値を使用することを示します。

しかし、実際に値がある場合はその値を使用するため、変数がnullになる可能性があることを認識し、ある場合は代替を提供します。

それで、getLocation内のコードに今学んだことを適用しましょう。したがって、これは特定の場合におそらく失敗する可能性のある行であることを知っています。たとえば、場所を取得する許可を得られなかった場合、またはユーザーが到達不能であり、GPSを把握できず、デバイスさえも把握できない場合ですGPSアドレスを知っています。

それでは、このコード行をラップしましょう。

そのため、Android Studioは読みやすくするために2行に配置しています。

しかし、これは実際にはすべて1行のコードであり、tryブロック内にラップします。

そこで、tryキーワードを追加し、括弧のセットを追加してから、tryブロック内のその位置に依存するすべてをラップします。次に、catchブロックを使用して、括弧のセット内でそれを追跡し、例外または単純にeを配置します。そして、catchブロック内で、エラーまたは例外が発生した場合に出力するか、より洗練されている場合は、実際にエラーの内容を把握し、それに基づいて代替ルートを決定できます。

したがって、ベンジャミン・フランクリンが準備に失敗したことで言うように、あなたは失敗する準備をしています。

そのため、何かが機能しないか、何かが例外をスローする可能性がある最悪のシナリオに常に備え、それを処理または処理するか、代替パスを提供できるように、それを試行してキャッチする必要があります。

それでは、try catchブロックのパズルの最後のピース、つまり実際に例外をトリガーするものについて話しましょう。

これらの例外は実際にはどこからスローされますか？これを理解する鍵は、throwキーワードです。

メソッドは例外をスローでき、例外をスローすることで、try catchブロックで例外を処理できます。

例外をスローするメソッドを作成する方法を紹介しましょう。

そこで、ここでは、somethingThatExpectsLessThan10という新しいメソッドを作成します。

これは本当に長いです。

そして、整数である入力を取る必要があり、それを呼び出します

n。メソッドの本体内で、この整数を使用して何かをする必要があります。

しかし、すべての状況で、nは常に10未満でなければなりません。また、nが10より大きい場合、何かひどいことが起こりました。

その場合、例外をスローする必要があります。

10を超えるものは使用できません。

そのため、この場合、throughキーワードを使用してエラーをスローし、「nは10より大きい、nは常に10未満でなければなりません」という例外を読み取ります。

だからここに例外がある。

そして、位置を出力してジオロケーターから位置を取得しようとする代わりに、getLocation内で、代わりにメソッドsomethingThatExpectsLessThan10を使用し、nに対して、明らかにルールに違反する12を入れます。動作しません。

したがって、runを押してコンソールをチェックアウトすると、「nは10より大きい、nは常に10未満でなければならない」という例外としてスローされることがわかります。

ただし、このtry and catchブロックがない場合に例外を処理せず、特定の条件下で例外をスローするこのコードを実行しようとすると、アプリが実際にクラッシュします。それでは、このメソッドをこことここで取り除きましょう。

そして、try catchブロックでコードを戻しましょう。 getCurrentPositionメソッドに戻り、このメソッドが実際に例外をスローする場所を示します。このために、geoロケーションパッケージ内を覗く必要があります。

このメソッドを使用し、COMMANDを押しながらクリックすると、宣言された場所または作成された場所に移動します。そして、ジオロケーターライブラリから作成されたこのメソッドで、アクセス許可が付与されたアクセス許可ステータスであるかどうかを確認するために確認できることがわかります。そのため、ユーザーは実際に自分の位置にアクセスすることを許可し、位置を取得しようとします。しかし、彼らがあなたに許可を与えなかった場合、無効な許可を処理します。

ここで、例外がどのようにスローされるかを確認します。

ジオロケーションパッケージによって例外がスローされる場合、許可が拒否される場合と許可が無効になる場合の2つの場合があります。

これは、例外の非動作をトリガーするthrowキーワードです。

これが、getCurrentPositionメソッドをtry catchブロックでラップした理由です。