最後のレッスンでは、AndroidとiOSの両方で現在位置を取得できるようにgeolocatorパッケージを実装しました。

そして、コードはとてもシンプルです。

基本的に、どの位置精度が必要かによって、電話の現在の位置を取得する1行です。

ただし、このコードで使用している2つのキーワードは、これまで実際に見たことのないパッケージのドキュメントに由来しています。

そのため、これを理解するには、まず非同期プログラミングとそれが有用な理由について学ぶ必要があります。

現在、コードを記述しているとき、実行に非常に短い時間がかかるコード行があります。

たとえば、printステートメントを実行する場合、1行のステートメントで、hello worldという単語がコンソールに出力されます。

そして、これには数秒で完了します。

しかし、時にはもっと時間がかかるかもしれないものがあります。

たとえば、インターネットから大きな画像をロードする場合。ブラウザで実行している場合でも、インターネットの速度がわずかに遅い場合でも、このような画像の読み込みが非常に長くかかることがあります。

これは、文字通りデータを取得しようとしている海中のケーブルを介して転送しようとしているデータがたくさんあるためです。また、データ量が多いほど、ロードにかかる時間が長くなります。この関数を作成したかどうかを検討してください。それは何も返さない関数です。したがって、voidを返しますが、3つのステップまたは実行しようとする3つのことで構成される本体を持っています。

そして最初のことは、単に「Hello moon」をコンソールに印刷することです。

これは非常に迅速に実行できます。

そのため、すぐに結果が表示されます。

次は、月のNASAから画像を読み込むことです。そして、あなたがどれくらいの頻度でこれを行ったかはわかりませんが、それらは巨大であり、長い時間がかかる可能性があります。

ただし、関数は上から下に進むため、3番目のステップは2番目のステップが完了するまで実行できません。

したがって、コンピュータが実際に最後のステートメントを実行するのは、このイメージがロードされた後だけです。つまり、「Hello Jupiter」という言葉をコンソールに出力します。

これで、この動作を同期と呼びます。

すべてが同期して起こります。

ステップ1が発生し、次にステップ2が発生します。ステップ2が終了または終了し、イメージが戻ってからステップ3が実際に実行されます。

これが非同期の場合はどうなりますか？

さて、手順1は通常どおり実行され、変更はありません。しかし、ステップ2では、NASAから非同期にイメージをロードしているとしましょう。

この場合、画像の読み込み中に、そのデータが戻ってくるのを待っている間に、既に手順3を実行できます。また、コードが続行する前にこれが完了するのを待つ必要はありません。

なぜこれが実際に役立つのでしょうか？

月曜日に出勤し、上司から100枚のパスポート番号を取得するように言われると想像してほしい。

今私にそれらを取得します。

したがって、すべての顧客を1つずつ呼び出して、パスポート番号を尋ねる必要があります。したがって、このチェックリストがあり、上から下へと順番に各顧客を呼び出しています。

今ではそれほど簡単ではありませんか？

頭の上のパスポート番号を覚えている人はほとんどいません。

そのため、電話をかけなければならないので、電話番号を尋ねた後、家の周りを調べて、実際にパスポートを見つけて、そのパスポート番号を渡さなければなりません。

そして、パスポートの隠し具合によっては、数秒から数分かかることがあります。

それで、あなたはそのすべてが起こっている間に何をしていますか？グラフを使用してこのプロセスを表現しましょう。横軸は時間であり、2、3の時間単位があります。各正方形を1秒として想像できます。

ここで、「お客様にパスポート番号を教えていただけますか？」

あなたの仕事は今ややや完了です。

次のステップは、ユーザーがパスポートを探してその番号を知らせるのを待っています。そして、その間ずっと、あなたはただそこに座って、彼らの親指をいじって電話をかけているだけです。そして、あなたは今本当に生産的ではありませんか？

しかし、パスポート番号が渡されると、ようやく行動に戻り、次の顧客に電話をかけることができます。その後、リストの最後まで行くと同じプロセスが何度も繰り返されます。 '100人の顧客に到達しました。そして、誰も電話を待っているのが好きではありませんか？

そして、それがあなたの一日の大半であるなら、それは素晴らしい気分にならないでしょう。

これを行う別の方法があることに気付いたかもしれません。

実際に、個々の電子メールを作成したばかりで、パスポート番号を顧客に説得してもらうために個人的なものにしたいと思ったとします。また、これらの電子メールをすべての顧客に送信することもできます。一つ。その後、これにより、必要な他のタスクを自由に実行できるようになります。

そして、返信が来たら、それらに返信できます。そのため、電話で応答を待つ必要がなくなります。

これを同じグラフィカルスタイルで表現する場合、あなたの一日はもっと似ているかもしれません。

各顧客に個別の電子メールを送信しますが、次のタスクに進む、次の電子メールを送信する、または次のことを行うことができ、彼らが時間をかけることができるため、顧客が応答するのにどれくらい時間がかかるかは実際には関係ありませんパスポートを見つけて、パスポート番号を取得すると、彼らはあなたに応答し、あなたはそのメールの返信を受け取ります。そして、その瞬間に実際に行動することができ、必要な情報を取得するのを待っている必要はありません。それで、あなたの一日はもっとこのように見えます。

生産性の高い何かをするために1日のすべての部分を使用しており、応答が来たときに送信した電子メールに応答しています。

この2つを並べて比較すると、ある場合には、何かをしている、待っている、何かをしている、待っている、次のことをしている、待っていることがわかります。

まあ、これは同期プログラミングです。他のケースでは、基本的にタスクを開始しており、そのタスクが完了すると通知を受け取り、それに基づいて行動します。

つまり、同じことを達成するために、9単位の時間ではなく5単位の時間しか使用できないということです。

これは、同期プログラミングと非同期プログラミングの違いです。

さて、これがすべて1人の作業者、つまりあなたによって行われていることに注目してください。 100人の顧客から100のパスポート番号を取得しようとする10人がいるわけではありません。実際にはまだ1人でそのタスクを実行していますが、顧客に時間をかけてそれらの番号を見つけさせ、結果が得られた後にのみ注意を引くようにするため、より効率的に実行されます。

それでは、実際の例を作成して、コードで記述した場合の外観を見てみましょう。クリマプロジェクト内で、右クリックして新しいDartファイルを作成します。そして、このファイルを単にスクラッチファイルと呼びます。

したがって、scratch.dartになり、メインプロジェクト内で作成されます。

このファイル内では、これはプロジェクトとは完全に分離されており、一部のDartコードをテストできます。

ただし、DartPadとは異なり、利用可能なすべてのFlutterおよびDartライブラリを利用できます。

このモジュールのコースリソースにあるこのリンクに移動すると、開始コードとして既に記述したコードが少しあることがわかります。

そこで、すべてをコピーして、scratch.dartに貼り付けます。

したがって、ここにあるのは、タスク1、タスク2、タスク3の3つの関数です。これらは、performTasksという関数内で順番に呼び出されます。そして、この特定のファイルへのエントリポイントであるメイン関数で、performTasksを呼び出します。

そのため、今すぐ、ファイルを右クリックして[scratch.dartを実行]をクリックするだけで、残りのコードを実行せずにこのscratch.dartを実行できます。コンソールがポップアップ表示され、これが分離されたファイルであり、UIをロードしてアプリにビルドするのに時間をかけることなく実行できることがわかります。

これが、Dartコードを試す場所です。

予想どおり、発生するのは、「タスク1が完了しました」、「タスク2が完了しました」、

タスク3は完了しました。」

しかし、注意すべき重要なことは、メソッド内で1、2、3の順にタスクを実行するため、これらの順序で発生することです。

これらの関数は順番に呼び出されます。

さて、タスク2は実際にはNASAのWebサイトから月の写真をダウンロードするような、本当に時間がかかるタスクだとしましょう。

さて、時間がかかるものをシミュレートしましょう。

threeSecondsという新しい期間オブジェクトを作成できます。

そして、これは新しい期間オブジェクトと等しくなります。

ここで、マイクロ秒、ミリ秒、秒、分、時間、日などの時間を指定できます。

この例を見るために、このビデオを何日も見続ける必要はありません。

それでは、3秒の遅延を設定するために、短くしておきますか？

したがって、この3秒間の遅延を実現できるようにするために、sleepという関数を使用します。

スリープに関する重要なことは、これが同期操作であることです。

そのため、次の行を実行する前に終了する必要があります。

また、期間オブジェクトを持つ単一の入力が必要であり、一時停止したい時間の長さとしてここにthreeSecondsを単純に入れることができます。

したがって、タスク2を呼び出すと、3秒の期間が設定され、プログラムが3秒間スリープします。そして、それから初めて、「タスク2完了」をコンソールに出力します。また、タスク3はタスク2の後に来るため、理論的には3秒遅れて来るはずです。

それでは、次に実行をクリックします。

したがって、現在実行するものとしてscratch.dartが選択されていることに注意してください。

これで、タスク1が完了し、1 2 3、タスク2タスク3が完了したことがわかります。したがって、このビデオを2倍速または2倍速で視聴している場合は、実際に通常の速度に戻すのが良いかもしれません。これらの遅延で実際に何が起こるかを説明するからです。

もう一度実行して、タスク1、1 2 3、タスク2タスク3に注目してください。

したがって、単一の文字列をコンソールに出力するこれらの各タスクは、基本的にごくわずかな時間しかかかりません。遅延を引き起こしているのは、dart：ioから来るスリープメソッドを使用してここに入れたこの人為的な遅延です。今では、物事をダウンロードしたり、ファイルから何かを読んだり、複雑な計算をしたりするなど、物事が予測できないほど長い時間を要する場合が多いため、すべてに多大な時間がかかります。

そして、私たちが起こしたくないのは、このタスクが完了するまで、他のすべて、後続のすべてのタスクを保留することです。

それで、私たちがそれを言うことができたら、あなたは何を知っていますか？タスク2をバックグラウンドで実行しますが、先に進み、今できることを実行します。

そして、その長いタスクが正常に完了したら、実際に発生した結果を処理します。

この場合、同期メソッドではなく非同期メソッドを使用します。

それでは、非同期遅延をシミュレートしましょう。

これを行うには、スリープを使用する代わりに、

Future.delayedというメソッドを使用できます。したがって、2つの入力があります。

1つは期間で、2つ目は遅延後に実行する計算です。

これが必要なものです。

したがって、最初の入力は3秒の遅延になります。次に、コールバックを追加して、これらの3秒が経過した後に何を行うかを指定します。つまり、この結果を作成し、タスク2が完了したことを印刷しています

それで、それを切り取って貼り付けます。

これで、コードの実行方法は非同期メソッドです。また、futureと呼ばれるものを返すため、非同期メソッドであることもわかっています。

そのため、ドキュメントで未来を見るたびに、メソッドは非同期的に発生するものであることがわかります。また、3秒遅延しますが、可能であればコードの他の行を実行できます。

これらの3秒が経過すると、コールバックで計算がトリガーされます。

この場合、結果を作成し、タスク2データと印刷タスク2を完了します。

コードを再度実行する場合は、コンソールを見て、今回はどのように機能するかを見てみましょう。

ここで、タスクを実行するという最上部では、その順序をまったく変更していないことに注意してください。

タスク1を実行する必要があり、タスク2、タスク3が呼び出されます。

しかし、ここで何が起こるか見てみましょう。

タスク1タスク3はほぼ即座に完了し、タスク2は3秒遅れて完了します。

したがって、これは非同期プログラミングの動作です。

これは、タスク2に時間がかかったためです。

それが起こるのを待っていたが、続けた。私たちは今できることを先に進めました。

そして、タスク2が完了すると、そのときだけ実際に実行され、コンソールに出力されます。

これはすべて、インターネットから画像をダウンロードしたり、時間のかかるタスクを実行したりする場合に非常に適しています。

ただし、次のタスクに進むためには、タスクの結果が必要な場合があります。

たとえば、タスク3が実際にタスク2からの入力を必要としたとしましょう。

タスク3を変更して、task2Dataという文字列入力を取得しましょう。次に、3の内部でそれを使用してみましょう。したがって、タスク3で完了し、この関数内にすべてのタスク2のデータを追加します。

現在、タスク3はタスク2からの何かに依存しています。

したがって、それを行うにはタスク2から何かを返す必要があります。

そこで、何もせずにタスク2から文字列を出力しましょう。

そして、ここでこの結果を出力します。

ここまでの結果と呼ばれる新しい文字列を作成しましょう。

そもそも何も設定せずに、非同期メソッドが3秒遅れた後にのみ設定しましょう。

そこで、インターネットからデータを取得しているとしましょう。完了したら、結果を値に割り当てます。

最後に、タスク2の出力として実際にこの結果を返しますか？

タスク2の出力ができたので、これをtask2Resultという新しい文字列に等しく割り当て、タスク2の出力に等しくすることができます。次に、タスク3内でそれを使用します。

したがって、タスク2の結果はタスク3の入力になります。

したがって、タスク2の全体の順序はタスク2が非同期で実行されるため、バックグラウンドで必要な限り実行できます。

しかし、それが完了したら、結果と呼ばれる文字列にデータを割り当て、完了したことを出力し、結果を出力します。

この段階で、その結果を取得して新しい文字列にバインドし、それをタスク3内の次のステップの入力として使用します。

タスク2が非同期であり、一度に完了することができ、コードが実際に先にスキップして今できないことを試してみることを知っているとしたら、どうなると思いますか？

したがって、コードの実行を見ると、タスク1とタスク3がほぼすぐに完了し、タスク2が完了するまでに3秒の遅延がかかっていることがわかります。ただし、タスク3はタスク2の一部のデータに依存しているため、タスク3がすぐに実行された場合、実際に渡す情報はありません。

そのため、タスク2が完了するまでtask2Dataは使用できないため、この時点でタスク3がnullで完了します。

では、どうすれば修正できますか？

実際にコードを実際に待機させることができるので、これは非同期メソッドであることがわかり、他のコードが続行できるようになり、これが完了するのを待つ必要がなくなります。

ただし、この場合、タスク3はこの2番目のタスクの完了に依存しているため、次のタスクを呼び出す前にコードが完了するのを待つことができます。これを行うには、これを非同期メソッドに変更します。

したがって、これは修飾子と呼ばれ、中括弧の直前に追加します。

その非同期キーワードを追加することにより、待機中のキーワードにアクセスできるようになりました。

そして、タスク2の前に配置します。

したがって、タスク2を呼び出す前に、タスク2が完了するのを待ちます。タスク2の内部で、いくつかの変更も行う必要があります。文字列を返す代わりに、futureと呼ばれるものを返す必要があります。

今、未来は未来に存在するものです。

現時点では、タスクが完了していないため何もありません。しかし、タスクが完了すると、将来は実際の文字列や実際の整数のような実際のものになります。ここで、このFuture.delayedメソッドが完了するのを待ってから結果を出力できるように、このメソッドを非同期としてマークする必要があります。

このように、非同期でこの更新されたコードを使用して、タスク3に入力を与えるために完了する必要のある部分を待機する場所で待機します。今度はplayを押してコードを実行すると、タスク1の完了が表示されます最初。そして、タスク2は遅れて起こり、最後にタスク3はタスク2の完了後に戻ってくるデータを使用して起こります。

デモでコードを書いているときに、先物について多くのことを話しました。先物とは正確には何ですか？

まあ、彼らは領収書に似ています。

意味を説明させてください。

コーヒーショップに行き、コーヒーを注文するとします。

ここで、注文番号が記載された領収書を受け取ります。コーヒーを飲みながら、離れて他のことをしたり、Instagramをチェックしたり、Facebookをチェックしたりできます。

あなたはそこに立ち、それが完了するのを待ちたくありません。

コーヒーが完成したら、「注文番号1で来てください」と注文を叫ぶでしょう。

これで表示され、注文の領収書に基づいて実際にコーヒーを請求できます。その領収書は、Dartの未来と同じです。

それはあなたが望んでいる実際のものではありません。

これは、将来何かを手に入れるという約束です。 JavaScriptを使用していて、約束を聞いたことがある場合は、まったく同じです。

現時点では存在しませんが、その処理が完了した後、画像を取得した後、またはテキストをダウンロードした後、実際の文字列や実整数などの実オブジェクトに未来が具体化されます。

これで、将来をより具体的にすることができます。

これが将来の文字列になるか、これが将来の整数になるかを指定できます。これを行うには、山かっこを使用します。

そのため、この場合、日付型を一般的な動的未来から特定のデータ型に変更できます。

したがって、将来の文字列になる可能性があります。

つまり、このメソッドをタスク2を呼び出すときに、将来の文字列を期待していることを意味します。そして今、この行をコメントアウトして、タスク2が何であるかを出力しようとすると、それがインスタンスであることがわかります将来の文字列。

まだ実際の文字列ではありません。タスク2が完了すると、将来の文字列が実現し、実際の文字列に変わります。現在、非同期プログラミングは把握するのが最も簡単な概念ではありません。しかし、私はこのレッスンを見直して再検討し、さらに多くの場所で非同期と待機を使用して先物を先取りするモジュールを完了することで、あなたが着実に慣れることを望んでいます非同期的に何かをするというこのアイデア。同期プログラミングであるすべてを順番に行うのではなく、プログラミングした順序で物事が発生しないように、タスクを非同期的に実行することにより、時間またはコンピューターの時間をより効率的にしようとしています、しかし、最短のルートを使用して、最速の方法でタスクを完了することになります。

したがって、このコンセプトは何度も登場します。

これがまだあなたの心の中に完全に収まっていなくても心配しないでください。これについては、今後さらに検討する予定です。ただし、次のレッスンでは、画面を読み込んだらすぐにアプリの作成と位置データの取得に戻ります。

それで、そのすべてについて、次のレッスンでお会いしましょう。