

XDKサンプルを含むコルーチン

# 概要

このサンプルは、XDKでコルーチンを使用する方法を示します。このコードには、コルーチンの技術仕様から次のものを含むいくつかの機能を実装しています:

* ベーシックフューチャー型
* ベーシック awaitableタイプ
* 既存の型で使用するためのco\_await演算子のオーバーロード

また、非同期プログラミングのためにC++ / WinRT基本ヘッダーで定義されている待てるタイプを使用する方法も示しています。

# サンプルの使用

コルーチンは、/ awaitコンパイラスイッチの使用を必要とするコンパイラ拡張です。Visual Studioを使用していると、これをプロジェクトのプロパティの「追加オプション」セクションに追加できます。コルーチンには、少なくともVisual Studio 2015 Update 3またはVisual Studio 2017が必要です。このサンプルは、Visual Studio 2017（15.5 update）もしくはそれ以降用に構築されており、またそこで最もよく機能します。

# サンプルの使用

このサンプルでは以下のアクションを使用できます。

|  |  |
| --- | --- |
| 作用 | ゲームパッドコントロール |
| アカウントピッカーを起動 | A | |
| コンソールで利用可能なユーザーを照会する | X | |
| 仮想キーボードを起動する | Y | |
| サンプルを終了する | 左トリガー + 右トリガー + Rボタン(rb) | |

各操作は、プログラムのフローを処理するために待てるタイプとコルーチンを使用するための異なるメカニズムを示します。

## オーバーロードされたco\_await演算子を使ったアカウントピッカーの処理

単項 **co\_await** 演算子はコルーチンの停止を要求するために使用されます。実行はコルーチンを呼び出した関数に返されます。この演算子は、**waitable** タイプと呼ばれる特定の型でのみ使用できます。Awaitable タイプには、以下のパブリックメンバー関数を含める必要があります。

* **await\_ready** - co\_await演算子を使用してコルーチンを停止するときに停止が必要かどうかを判断するために呼び出されます。操作が予想より早く終了した場合、この関数は**true**を返して停止する必要がないことを通知することができます。又は、コルーチンを停止するために**false**を返すことができます。
* **await\_suspend** - コルーチンが停止されているときにこの関数が呼び出されます。**await\_ready**が**true**を返した場合、 この関数は呼び出されません。例えばコルーチンが停止される前にawaitableオブジェクトに情報を格納するなど、必要な操作を処理するために使用できます。
* **await\_resume** - コルーチンが再開されると、この関数はco\_await演算子で返される値を決定します。

アカウントピッカーを開くAPIは **IAsyncOperation <AccountPickerResult ^> ^** オブジェクトですが、これはawaitable タイプではありません。グローバル **演算子co\_await** オーバーロードは、このタイプのオブジェクトでco\_await演算子を使用できるようにするために、クライアントコードによって提供されています。停止ロジックを処理するために**accountpicker\_await\_adapter**と言う代替オブジェクトを提供します。、それは一時停止ロジックを扱うawaitableです。**await\_ready** 関数は**IAsyncOperation**のステータスをチェックします ;すでに完了している場合は、コルーチンを停止する必要はなく、完了するまで実行できます。**await\_suspend** 関数は、 IAsyncOperationの**Completed**イベントを使用して、アカウントピッカーが閉じられたらコルーチンを継続するための新しいコンテキストを作成します。このサンプルの**accountpicker\_await\_adapter**の実装 は XDKに同梱されているC ++ / WinRTプロジェクションヘッダーで定義されている**winrt::accountpicker\_await\_adapter**構造体に基づいています。

## カスタムのfuture型の仮想キーボードを扱う

コルーチンのシグネチャは他のサブルーチンと同じように見えます。戻り型、名前、および引数リストがあります。唯一の違いは、戻り型が **future** タイプ。**std::future**タイプを使用でき、ほとんどの目的には十分です。このサンプルは**awaitable\_future**と言う基本的なfuture型を実装して、 他の実装の参考として使うことができます。awaitableタイプと異なり、future型は特定のパブリックインタフェースを必要としません。代わりに、future型は自身のスコープ内で '**promise\_type**'という名前の構造体を提供しなければなりません。promise\_typeは、次のpublic インタフェースを提供しなければなりません。

* **get\_return\_object** - この関数は、コルーチンが呼び出されたときに呼び出されますが、それはコルーチンの本体が実行を開始する前のこととなります。コルーチンが停止された場合に呼び出し元に返されるfuture オブジェクトの構築を担当しています。
* **（オプション）get\_return\_object\_on\_allocation\_failure** - この関数の提供によって、コルーチンコンテキストが呼び出されたときに使用可能なメモリがない場合でも適切な処理が可能となります。
* **initial\_suspend** - コルーチンの本体が、即座に実行を停止して呼び出し元に返すべきかどうかの判断の実行開始する前に呼び出されます。この関数が**true**を返すと、 コルーチンが実行を開始します。**false**が返された場合、コルーチンは実行の開始する前に一時停止しますが、それはfutureオブジェクトが作成された後となります。
* **final\_suspend** - これは、実行がコルーチンの最後まで行われた時、または **co\_return** ステートメントに遭遇したときに呼び出されます。この関数が**true**を返すと、コルーチンコンテキストは含まれる全データを無効にするpromise\_typeとともに、すぐに解放されます。この関数が**false**を返すと promise\_typeによって保持されているデータは、呼び出し元が照会するのに有効なままですが、呼び出し元はコルーチンコンテキストのクリーンアップにも責任を持ちます。
* **return\_value** または **return\_void** - promise\_typeはこれらの機能を1つのみ実装しなければなりません。コルーチンが **co\_return** ステートメントを使用して呼び出し元に値を返す場合その値は、 **return\_value** に渡され呼び出し元が照会するためのpromise\_typeに格納されます。co\_returnステートメントがない場合、または値のないco\_returnステートメントがある場合には、 **return\_void** を実装する必要があります。どちらの場合でも、コルーチンの実行が終了すると、関数は**final\_suspend** の直前に呼び出されます。
* **（オプション）operator new** - これがpromise\_typeで提供されている場合は、**::operator new**の代わりにコルーチンコンテキストにメモリを割り当てるために使用されます。 。これはメモリ使用量を追跡したり、最適化されたアロケータを提供するのに役立ちます。
* **（オプション）operator delete** - もしもオーバーロードされたオペレータnewが提供されている場合は、コロティクコンテキストが解放された後にメモリを解放するためにこれを提供する必要があります。

**accountpicker\_await\_adapter**とは異なり、 、クライアントコードによって提供される**keyboard\_await\_adapter** は非同期TCUI機能が完了すると自動的に再開されません。代わりに、適切な時間にコルーチンを再開するという事は呼び出し元への依存となっています。

## 同じ関数スコープ内に複数のスレッドを持つユーザーコレクションのクエリ

ユーザーへの問い合わせ::レンダリングスレッドへのアクセスである場合、ユーザーは非常に時間がかかり、ヒッチの原因となりえます。C++ / WinRTプロジェクションヘッダーには、C ++/ WinRTプロジェクションを利用しないゲームであっても、単一の関数内でスレッドをシームレスに処理するために使うことができる多くの適切な型が含まれています。co\_await演算子を持つ **resume\_background** オブジェクトのインスタンスを使用することによって、コルーチンを呼び出した関数がただちに実行を再開し、コルーチンの実行の即座の続行のためにも新しいスレッドが作成されます。これはUser::Usersの、コルーチン本体の他の部分と同一のスコープ内のバックグラウンドスレッドへのアクセスをオフロードします。他に用意されているawaitable 型のソースコードは、XDKに同梱されているC ++/ WinRT投影ヘッダーのbase.hにあります。

# 更新履歴

2018年2月 - 初回リリース

# プライバシーステートメント

サンプルをコンパイルして実行すると、サンプルの実行可能ファイルの名前がMicrosoftに送信され、サンプルの使用状況の追跡に役立ちます。このデータ収集を無効にするには、Main.cppの「Sample Usage Telemetry」というラベルの付いたコードブロックを削除します。

マイクロソフトのプライバシーポリシー全般に関する詳細については、 [Microsoftのプライバシーステートメント](https://privacy.microsoft.com/ja-jp/privacystatement/)をご参照ください。