

SprCSharp における例外の取り扱い

C++で作成された Springhead library を C#側から呼び出す場合、library 内で発生する例外の情報を C#側で取得するには次のような困難がある。

- (1) library 内で発生する “access violation”, “array bounds exceeded” などの例外は Windows では Structured Exception として扱われ、通常の C++における try ~ catch の仕組みでは捉えることができない。
- (2) C++のコードは unmanaged space 上で実行されるため、library 内で発生した例外を自動的に unmanaged-managed 境界を超えて伝搬させることができない。

このため、次のような取り扱いを実装する。

Structured Exception の扱い

- a) Structured Exception は、プログラム実行時に `_se_translator_function` 型のハンドラ SEHandler (`se_translator()`) を登録することで捉えることができる (このコードは/EHa オプション付きでビルドする必要がある)。
- b) SEHandler の中で `std::exception` を継承した例外クラス SEH_Exception のインスタンスを生成して throw することにより、通常の try ~ catch コードで捕捉できる例外にすぐ替えることができる。
- c) SEHandler は、例外が発生した try 節に対応する catch 節よりも前に呼び出されるため Springhead library の中でこれを捉えることも可能であるが、library のコード自体に変更を加えることは望ましくないため、そこでは catch せず、それを呼び出すアダプタメソッド (SprExport.dll) の中で捉えるものとする。
- d) アダプタメソッドにおいて例外を補足した時には、例外を起こした library 関数からアダプタメソッドまでの call stack は既に巻き戻されてしまっているため、必要な trace back 情報を得ることができない。このため SEHandler が呼び出された時点 (SEHandler は例外が発生したら最初に呼び出されるため、例外発生時点での call stack がまだ残されている) で trace back 情報を取得しておき、後でこの情報を取り出す仕組みを作ることとする。SEHandler は複数回呼び出される可能性があるので、2 回目以降は何もしないようにする。

unmanaged-managed 境界を超えての例外伝搬

- a) C#側に SEH_Exception を継承した例外クラス SEH_Exception を用意し、“このクラスの例外を throw するメソッド RaiseException() を呼び出す C++関数 raise_managed_exception()” を SprExport.dll 内 (unmanaged space) に登録する仕組みを用意する。

- b) Springhead library 内で例外が発生し SprExport.dll 内でそれを catch したならば、登録されている `raise_managed_exception()` を呼び出す。
- c) `raise_managed_exception()` は C#側 (managed space) で次のことを実行する。
 - (a) SEHandler(C++側) が作成した trace back 情報を取り出す。
 - (b) `System.Exception` のインスタンスを生成して throw する。(a) で取り出した trace back 情報を例外情報とする。

以上の事項を次葉の Fig 1 に図示し、その後に説明を加える。

関連するソースファイルは、

`SprCSharp/CSUtility.cs`

`SprImport/CSUtility.cs`

`SprExport/CSUtility.h`

`SprExport/CSutility.cpp`

である。

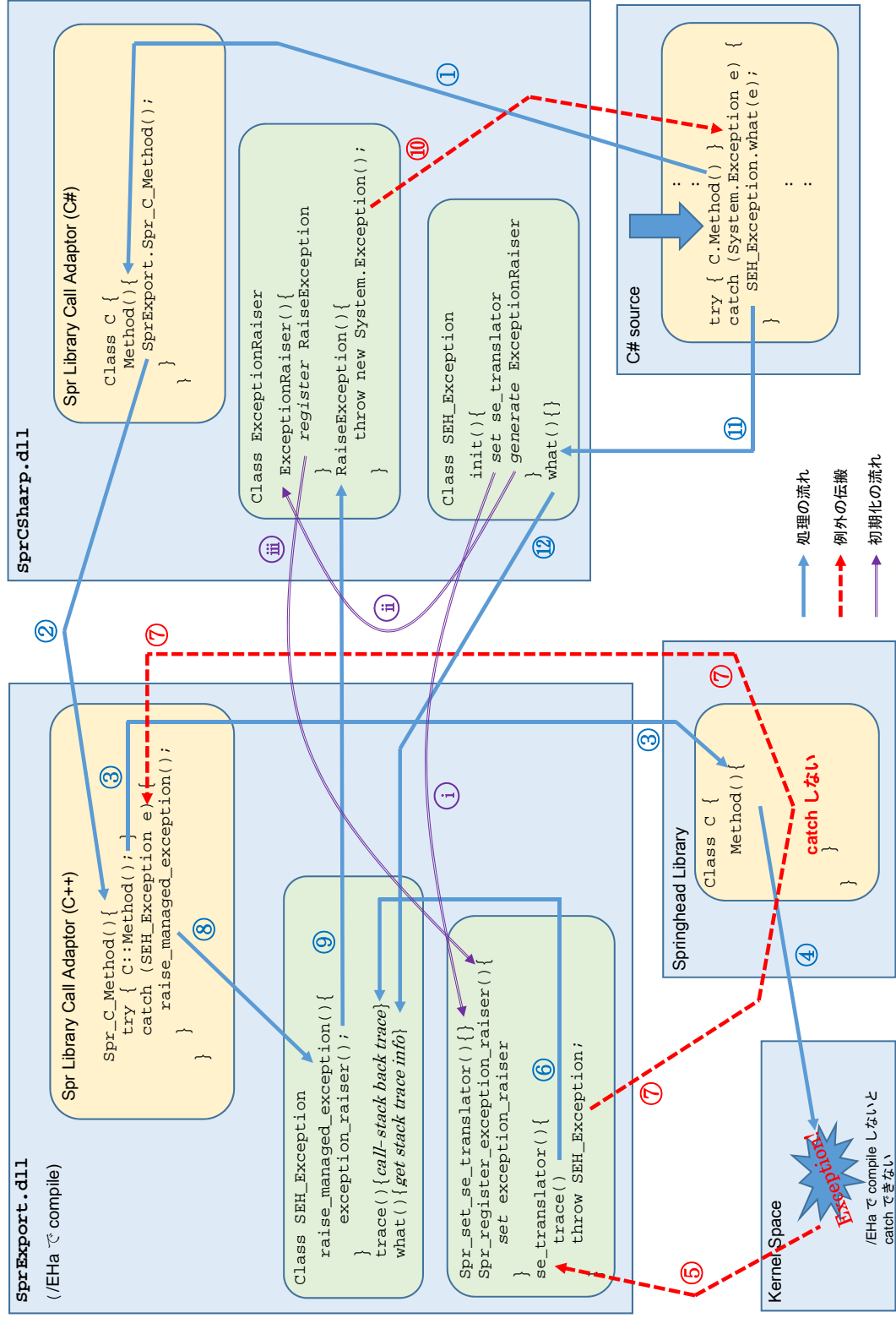


Fig 1 例外の伝搬

前葉の図に従って処理及び例外の流れを説明する。項番は図中の矢線番号に対応する。

初期化の流れ

- i) unmanaged space (SprExport.dll) に SEHandler を登録する。
C# SEH_Exception.init()
C++ Spr_set_se_translator()
- ii) クラス ExceptionRaiser のインスタンスを生成する。
C# EXceptionRaiser.ExceptionRaiser()
- iii) それを unmanaged space に登録する。
C++ Spr_register_exception_raiser()

処理/例外の流れ

- 1) ユーザソースから C#側 (SprCSharp.dll) で定義されたクラス C のメソッドを呼び出す。
C# C.Method()
- 2) C.Method()(managed) から SprExport.dll(C++, unmanaged) で定義されている対応メソッドを呼び出す。
C# SprExport.Spr_C_Method()
C++ Spr_C_Method()
- 3) Spr_C_Method() から Springhead library 中のメソッドが呼び出される。
C++ C::Method()
- 4) このメソッドの中で Structured Exception が発生する。
- 5) SEHandler が呼び出される。
C++ se_translator()
- 6) ここで call stack の trace back を行なう。
C++ SEH_Exception::trace()
- 7) 続けて SEH_Exception のインスタンスを生成して throw する。Springhead library の中ではこの例外は catch せず、呼出し階層を遡って SprExport.dll の中で catch する。
C++ Spr_C_Method()
- 8) C#側に例外を伝搬するためのシーケンスを開始する。
C++ SEH_Exception::raise_managed_exception()
- 9) unmanaged から managed を呼び出して、C#の例外を発生させる。
C# ExceptionRaiser.RaiseException()
C++ SEH_Exception::raise_managed_exception()
- 10) これをユーザソースで System.Exception として catch する。

- 11) catch した例外インスタンスを引数として stack trace 情報を取得する。ここで catch した例外インスタンスは C#側の call stack の情報しかもっていない。

```
C# SEH_Exception.what()
```

- 12) unmanaged 側で取得しておいた stack trace 情報を取り出し、引数で渡された C#側の stack trace 情報とマージすることで全 stack trace 情報を作成する。

```
C# SEH_Exception.what()
```

```
C++ SEH_Exception::what()
```

最後に、Unity から Springhead library を呼び出す場合の注意点を記す。

- 例外伝搬機構の初期化を 1 回だけ行なう必要がある。

```
using SprCS;
```

```
SEH_Exception.init();
```

PHSceneBehaviour.cs の Build() の冒頭で初期化している。

- 例外は System.Exception として補足する。stack trace は、補足した例外インスタンスを引数としてクラス SEH_Exception のメソッド what() を呼び出すことで取得する。

```
catch (System.Exception e) {  
    Debug.Log(SEH_Exception.what(e));  
}
```

- Unity の console タブにおいて “Error Pause” を on にしておかないと、例外発生と同時に Unity が終了してしまうので注意が必要である。

【補足】例外処理のログ

実行時に環境変数 SRPCS_LOGFILE にファイルパスを設定しておけば、そこで指定されたファイルに "CSUtility.cpp" で例外を処理した状況を出力することができる。

ログのサンプル (一部折り返し)

```
2106/07/25 11:51:49 [06088] DllMain: DLL_PROCESS_ATTACH: DLL is loaded dynamically.
```

```
2106/07/25 11:51:49 [06088] --- set SEH translator (0x105BCAB0).
```

```
2106/07/25 11:51:49 [06088] ** managed-exception raiser registered **
```

```
2106/07/25 11:51:50 [06088] ** enter: se_translator **
```

```
2106/07/25 11:51:50 [06088]
```

```
SEH_Exception: EXCEPTION_ACCESS_VIOLATION at 0x0FD6F58A
```

```
Reading from address 0x00000005
```

```
stack trace:
```

```
0x001DE4D8:
```

```
[0] 0x0FD73539 SprExport.dll: Spr::ObjectIf::AddRef
```

```
f:\project\springhead2\src\foundation\foundationstub.cpp: 217
```

```
[1] 0x0FD31BD7 SprExport.dll: Spr::UTRef<Spr::PHSceneIf>::UTRef<Spr::PHSceneIf>
```

```
f:\project\springhead2\include\base\baseutility.h: 132
```

```
[2] 0x1029ABFF SprExport.dll: Spr::PHSdk::AddChildObject
```

```
f:\project\springhead2\src\physics\phsdk.cpp: 187
[3] 0x1029A92E SprExport.dll: Spr::PHSdk::CreateShape
      f:\project\springhead2\src\physics\phsdk.cpp: 155
[4] 0x1016CF59 SprExport.dll: Spr::PHSdkIf::CreateShape
      f:\project\springhead2\src\physics\physicsstub.cpp: 3735
[5] 0x10538C4D SprExport.dll: Spr_PHSdkIf_CreateShape
      f:\project\springhead2\src\sprcsharp\sprexport\csphysics.cpp: 11768
2106/07/25 11:51:50 [06088] ** leave: se_translator **
2106/07/25 11:51:56 [06088] DllMain: DLL_PROCESS_DETACH:
      DLL will be released by process termination.
2106/07/25 11:51:56 [06088] - unset SEH translator (0x00000000).
```