難読化後のよくあるエラーと解決方法

難読化はソフトウェアのセキュリティを高めるための重要な機能ですが、難読化後に動作しなくなることがあるため、ツールの使用を避ける人もいるかもしれません。
しかし、このような動作不良には「理由」が存在します。ここでは、よく発生する 2 つのエラーのケースと、その解決策を紹介します。 難読化後のスタックトレースを元の名前に戻す方法についても説明します。

目次

1）ケース１（リフレクション）

リフレクションを使用して動的にクラスやメソッドを参照するコードでは、難読化により名前が変更されるため、エラーが発生することがあります。
コード例

```
#nullable disable
using System;
using System.Reflection;

class Program
{
    static void Main(string[] args)
    {
        try
        {
            string className = "SampleClass"; // 難読化で変更される
            string methodName = "SampleMethod"; // 難読化で変更される

            Type type = Type.GetType(className); // 難読化後は null を返す（エラー発生: 行 10）
            if (type == null)
            {
                Console.WriteLine("クラスが見つかりません。難読化の影響かもしれません。");
                return;
            }

            MethodInfo method = type.GetMethod(methodName); // 難読化後は null を返す可能性あり（エラー発生: 行 14）
            if (method == null)
            {
                Console.WriteLine("メソッドが見つかりません。難読化の影響かもしれません。");
```

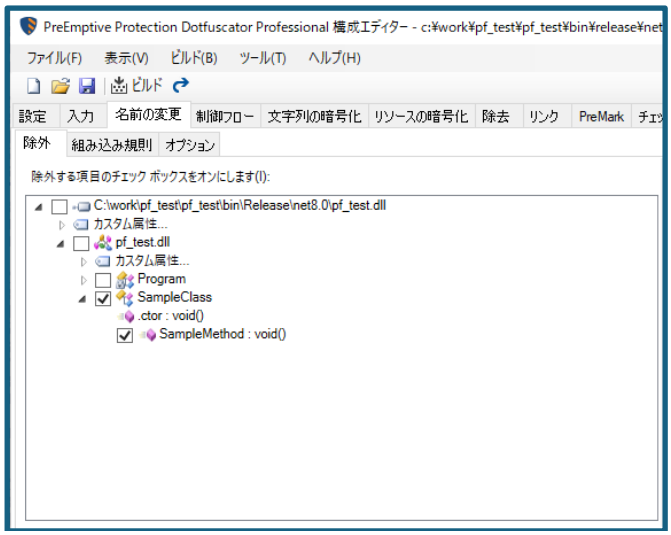
```
            return;
        }

        object instance = Activator.CreateInstance(type);
        method.Invoke(instance, null);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"エラー: {ex.Message}");
    }
  }
}

public class SampleClass
{
  public void SampleMethod()
  {
    Console.WriteLine("リフレクションでメソッドが呼び出されました。");
  }
}
```

エラーの詳細

1. 行 10: Type.GetType(className) が null を返す。
    - 理由: 難読化でクラス名が変更されているため。
2. 行 14: type.GetMethod(methodName) が null を返す。
    - 理由: 難読化でメソッド名が変更されているため。

解決策

ツールの設定で、特定のクラスやメソッドの名前変更を除外します。 今回、SampleClass と SampleMethod の 2 つを除外する必要があります。以下の画面キャプチャを参照してください。



---

2) ケース 2（JSON シリアライザ）

JSON シリアライザは、プロパティの名前が変更されると正しくデシリアライズできないことがあります。

コード例

```
using System;
using System.Text.JSON;

class Program
{
   static void Main(string[] args)
   {
      string JSON = "{¥"Name¥":¥"Test¥",¥"Age¥":25}"; // 正しいキーの例

      try
      {
         var obj = JSONSerializer.Deserialize<Person>(JSON); // エラー発生の可能性: 行 8
         if (string.IsNullOrEmpty(obj.Name) || obj.Age == 0)
         {
            throw new InvalidOperationException("必要プロパティが設定されていません！");
         }

         Console.WriteLine($"名前: {obj.Name}, 年齢: {obj.Age}");
      }
      catch (Exception ex)
      {
         Console.WriteLine($"エラー: {ex.Message}");
      }
   }
}
```

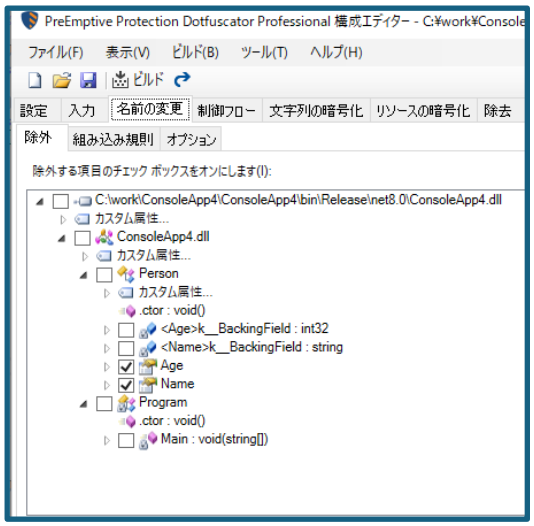
```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

エラーの詳細

1. 行 8: JSONSerializer.Deserialize<Person>(JSON) が正しく動作しない。
    - 理由: 難読化でプロパティ名が変更されているため、JSON のキーと一致しなくなる。

解決策

シリアライズの対象となるクラスの名前を変更しないよう設定します。 今回、Age プロパティと Name プロパティを除外する必要があります。以下の画面キャプチャを参照してください。



---

3) 難読化後のスタックトレースを元に戻す方法

難読化されたスタックトレースを元の名前に戻すことで、デバッグの効率が改善します。

コード例

```
using System;

namespace ObfuscationTest
{
    public class Program
    {
```

```
    public static void Main(string[] args)
    {
        try
        {
            var testClass = new TestClass();
            testClass.MethodThatThrowsException(); // 例外をスロー (エラー発生: 行 8)
        }
        catch (Exception ex)
        {
            Console.WriteLine("エラーが発生しました:");
            Console.WriteLine(ex.ToString());
        }
    }
}

public class TestClass
{
    public void MethodThatThrowsException()
    {
        // 意図的に例外を発生させる
        throw new InvalidOperationException("これは難読化後のテスト例外です。");
    }
}
}
```

難読化前後のスタックトレース例

1. 難読化前:

```
System.InvalidOperationException: これは難読化後のテスト例外です。
  at ObfuscationTest.TestClass.MethodThatThrowsException() in C:¥work¥ConsoleApp5¥Program.cs:line 27
  at ObfuscationTest.Program.Main(String[] args) in C:¥work¥ConsoleApp5¥Program.cs:line 12
```

2. 難読化後:

```
System.InvalidOperationException: これは難読化後のテスト例外です。
  at b.a() in C:¥work¥ConsoleApp5¥Program.cs:line 27
  at a.a(String[] A_0) in C:¥work¥ConsoleApp5¥Program.cs:line 12
```

難読化後のスタックトレースは、クラス、メソッド名が難読化されており、デバッグが困難です。

解決策

Lucidator を使用して、スタックトレースを元に戻します。 Lucidator は、NuGet パッケージとして配布されています。

以下のページを参考にインストールすることができます。

https://www.preemptive.com/dotfuscator/pro/userguide/ja/interfaces_lucidator.html

Lucidator は、`lucidator [オプション] [割り当てファイル] [スタックトレースファイル]` の形式で使用します。
主なオプションは以下の通りです：
- `/mapfile=<割り当てファイル>`: 割り当てファイルを指定（例: `/mapfile=map.xml`）
- `/stacktracefile=<スタックトレースファイル>`: スタックトレースファイルを指定（例: `/stacktracefile=stacktrace.txt`）

実行例:

```
lucidator.cmd /mapfile=C:¥work¥ConsoleApp5¥bin¥Release¥net8.0¥Dotfuscated¥Map.xml
/stacktracefile=C:¥work¥ConsoleApp5¥bin¥Release¥net8.0¥Dotfuscated¥stack.txt
戻されたスタックトレース:
System.InvalidOperationException: これは難読化後のテスト例外です。
  at ObfuscationTest.TestClass.MethodThatThrowsException() in C:¥work¥ConsoleApp5¥Program.cs:line 27
  at ObfuscationTest.Program.Main(string[]) in C:¥work¥ConsoleApp5¥Program.cs:line 12
```