

A hand-drawn orange smiley face is painted on a dark asphalt surface. The smiley face consists of a circular outline and two dots for eyes. The text "Null Safety" is written in a bold, black, sans-serif font across the center of the smiley face. Below the smiley face, the words "STAY SAFE" are painted in white, uppercase letters on the asphalt.

Null Safety

STAY SAFE

2023年10月25日(水)

アジェンダ

1. Nullとは？
2. Null Safetyとは？
3. 試してみた
4. まとめ

NULLとは

プログラムの世界では、
「**何もない**」ということを表します。
“0”ではなく、“何もない”



1



0



NULL

よく見るやつ、NULL参照

例) Java

Java.lang.NullPointerException

→値がnullの変数を参照した時に発生する例外

^ ^
(^v) < ぬるぽ

(.v.) || ガッ
と) ||
Y /) ^
/) < > ^ n
L / / .v D) / ←>> 1
[フシ /

よく見るやつ、NULL参照

nullの問題点とは

“ 1965年に考案したnull参照の概念は、10億ドル単位の過ちと呼ぶべきものであろう。これこそがその後40年に渡り、数え切れないほどのエラーや脆弱性、システムクラッシュの原因となり、10億ドル単位の損害や苦勞を引き起こしてきたのである。

という言葉が飛び出すほど、影響を及ぼしてきた

引用： <https://genesis-tech.jp/blog/null-safety-of-csharp/>

NULLへの対策

```
1 | var person = TryGetPerson(id);  
2 | string name = person.Name;
```

^ ^
(^v) < ぬるぽ



```
1 | var person = TryGetPerson(id);  
2 | if( person == null )  
3 |     return;  
4 | string name = person.Name;
```

変数・personがnullで返却されてきた場合、personのNameにアクセスしようとする、いわゆる「null参照」で落ちることになる。

回避するにはnull判定のコードを書く必要がある。

上記のように、様々な場面でnullを想定して回避する実装をすれば対処できるが、見落としなどにより、**nullチェックができていない実装**や、**nullが来ることを期待していないのに、nullが来てしまう実装**が存在すると、null参照で落ちる危険性がある。

引用：<https://genesis-tech.jp/blog/null-safety-of-csharp/>

**このNULLにつきまとう問題を解決するため、
先人たちの努力により、新たな仕組みが作られました。**

Null Safety とは

※C#を例に

**NULLが原因で発生するエラーをコンパイル時に防止する仕組み
NULL安全 と呼ばれる**

→NULLを扱える変数、扱えない変数を区別して
コンパイル時からNULLが原因で発生するエラーを防止する



参考「null許容値型」と「null参照許容型」

もともと、C#では値型（intやstructなど）はnull値が取れず、
値型の変数でnull値を代入するには、**「null許容値型」**を使う必要があった。

(int? のように ? を付けて宣言)

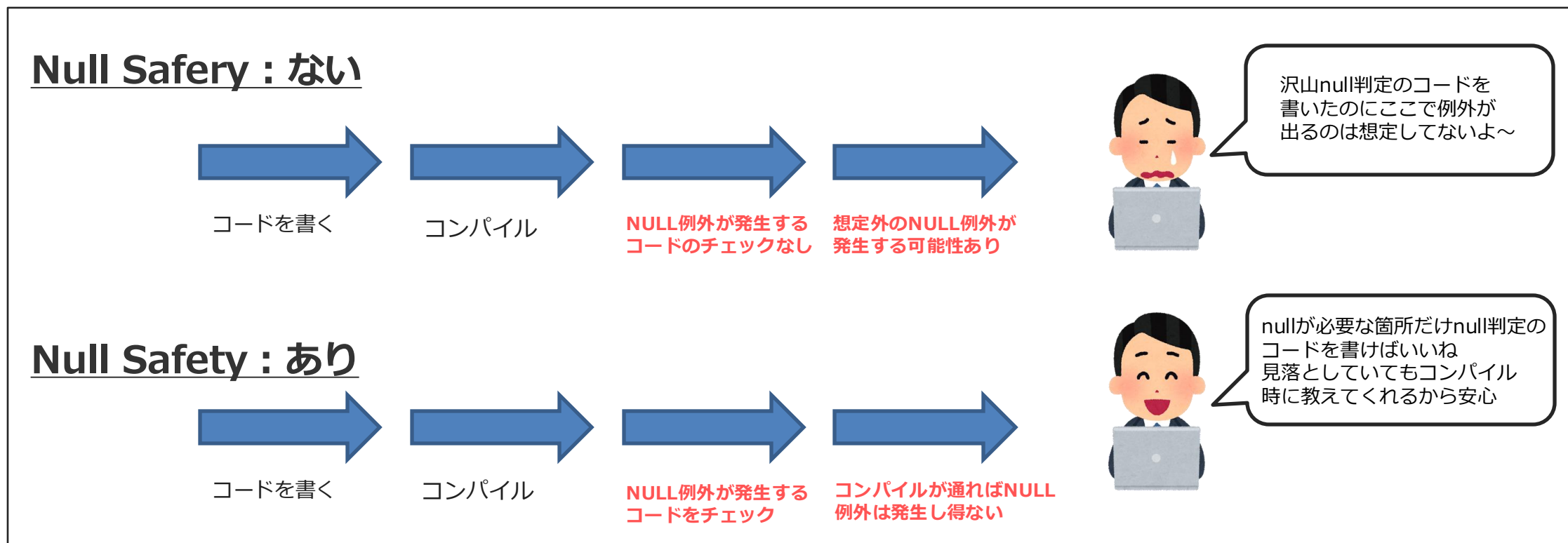
一方、参照型（stringやobjectなど）は、**デフォルトでnullが許容されているので、
許容・非許容が区別されていなかった。**

「null参照許容型」の登場により、C#でも**参照型の「null許容 / null非許容」**の
区別が付くようになったので、null参照例外が発生するコードは、
コンパイル時に検出してくれるようになり「null安全」が満たされるようになった。

引用： <https://genesis-tech.jp/blog/null-safety-of-csharp/>

よいところ

- ・ 想定外の箇所で変数がnullになり、不具合になってしまうことを防げる。
- ・ nullの可能性を想定して、nullを判定するコードを書く必要がなくなる。
(nullが必要な箇所は区別して、nullを判定するコードを書く。)



NULL SAFETY の使い方

→ 「null参照許容型」 を有効にする ※C# 8.0から導入

有効にすると、参照型のTに対して、**nullを認めなくなる**(null非許容 → non-null)
このため、nullを設定できない型にnullを設定しているコードは、
コンパイル時に検出されるようになった。

```

C:\Users\vega\source\repos\Project1\Project1\Project1.csproj - sakura 2.4.1.2849
ファイル(F) 編集(E) 変換(C) 検索(S) ツール(T) 設定(O) ウィンドウ(W) ヘルプ(H)
1 <?xml version="1.0" encoding="utf-8"?>
2 <Project ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/develop
3 <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Micr
4 <PropertyGroup>
5 <Configuration Condition="'$(Configuration)' == ''">Debug</Config
6 <Platform Condition="'$(Platform)' == ''">AnyCPU</Platform>
7 <ProjectGuid>{4804A292-BBE4-4603-BE55-B9E77A933B02}</ProjectGuid>
8 <OutputType>Exe</OutputType>
9 <RootNamespace>Project1</RootNamespace>
10 <AssemblyName>Project1</AssemblyName>
11 <TargetFrameworkVersion>v4.7.2</TargetFrameworkVersion>
12 <FileAlignment>512</FileAlignment>
13 <AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
14 <Deterministic>true</Deterministic>
15 <LangVersion>8.0</LangVersion>

```

```

#nullable enable
object obj2 = null; // warning CS8600 Null リテラルまたは Null の可能性がある値を Null 非許容型に変換しています。
string s1 = null; // warning CS8600 Null リテラルまたは Null の可能性がある値を Null 非許容型に変換しています。
int i1 = null; // error CS0037: Null 非許容の値型であるため、Null を 'int' に変換できません。

```

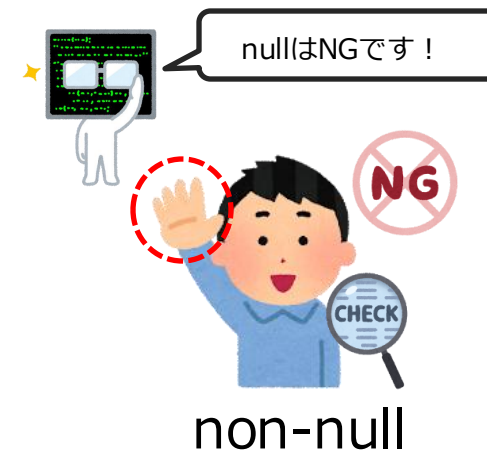
修正しなきゃ

```

#nullable enable
object obj2 = "hoge";
string s1 = "hogeta";
int i1 = 1;

```

コンパイル時に検出される



参考：<https://ufcpp.net/study/csharp/resource/nullablereferencetype/>

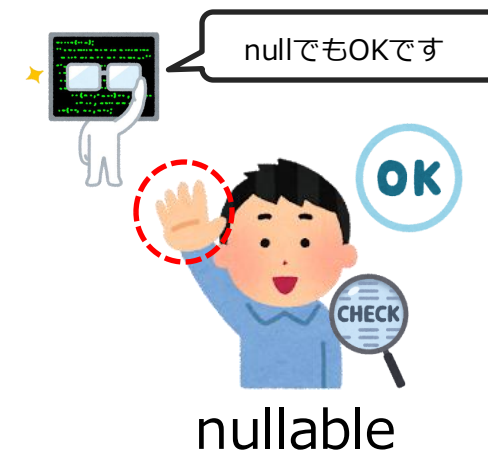
NULL SAFETY の使い方

仕様や運用上で、nullを扱う場合には、**T?** の形で記述することで、**nullを許容することができる**(null許容 → nullable)
※nullを許容してしまうので、従来通りnullチェックの構文は書く必要がある。

```
#nullable enable  
object? obj3 = null;  
string? s2 = null;  
int? i2 = null;
```



コンパイルエラーにならない



■:警告 ■:エラー ■:OK

```
// ※#のバージョンは8.0以上しておくこと
object? obj1 = null; // warning CS8632: '#nullable' 注釈コンテキスト内のコードでのみ、Null 許容参照型の注釈を使用する必要があります。
#nullable enable
object obj2 = null; // warning CS8600 Null リテラルまたは Null の可能性がある値を Null 非許容型に変換しています。
object? obj3 = null;

string s1 = null; // warning CS8600 Null リテラルまたは Null の可能性がある値を Null 非許容型に変換しています。
string? s2 = null;

int i1 = null; // error CS0037: Null 非許容の値型であるため、Null を 'int' に変換できません。
int? i2 = null;
```

ディレクティブ外で "?" を定義すると警告が出る

#nullable enable

ディレクティブ



コンパイル

```
ビルドを開始しました...
1>----- ビルド開始: プロジェクト: Project1, 構成: Debug Any CPU -----
1>C:\Users\vega\source\repos\Project1\Project1\NullSafety.cs(18,19,18,20): warning CS8632: '#nullable' 注釈コンテキスト内のコードでのみ、Null 許容参照型の注釈を使用する必要があります。
1>C:\Users\vega\source\repos\Project1\Project1\NullSafety.cs(29,22,29,26): error CS0037: Null 非許容の値型であるため、Null を 'int' に変換できません
1>C:\Users\vega\source\repos\Project1\Project1\NullSafety.cs(21,27,21,31): warning CS8600: Null リテラルまたは Null の可能性がある値を Null 非許容型に変換しています。
1>C:\Users\vega\source\repos\Project1\Project1\NullSafety.cs(25,25,25,29): warning CS8600: Null リテラルまたは Null の可能性がある値を Null 非許容型に変換しています。
===== ビルド: 成功 0、失敗 1、最新の状態 0、スキップ 0 =====
===== ビルドは 5:06 PM に開始され、00.312 秒 かかりました =====
```

下記の条件演算子や、合体演算子を組み合わせて使う

```
//null条件演算子 ns が null の場合は name1 は null。 null ではないは  
var ns = new Ns();  
ns.name = null;  
var name1 = ns?.name;  
  
// null合体演算子 personNameがnullの場合、"is null 1" を代入する。  
string personName = null;  
var name2 = personName ?? "is null 1";  
  
// null条件演算子 + null合体演算子 personNameがnullの場合、"is null  
var name3 = ns?.name ?? "is null 2";
```

参考：<https://atmarkit.itmedia.co.jp/ait/articles/1607/07/news023.html>

まとめ

- NULLとは、“0”ではなく“何もない”
- NULLは昔から影響を及ぼしてきた

★Null Safetyは、NULLが原因で発生するエラーをコンパイル時に防止する仕組み

- NULLを扱える変数、扱えない変数を区別してプログラムの安全性を高める。
(null許容型とnull非許容型)
- 想定外の箇所で変数がnullになり、不具合になってしまうことを防げる。
- nullの可能性を想定して、nullを判定するコードを書く必要がなくなる。
(nullが必要な箇所は区別して、nullを判定するコードを書くだけでよい。)

→NULLになり得る状況を見逃さないようにして、そのための記述もシンプルに書きやすく！

