Understanding Nullable and Non-Nullable Reference Types

Jason Roberts
.NET DEVELOPER

@robertsjason dontcodetired.com

Overview

An overview of C# 8+ null features

Enable non-nullable reference types

Specifying a reference should be nullable

Non-nullable properties

Non-nullable method return values

Null-coalescing and null-conditional operators

The null-forgiving operator

Refactoring existing code

Considerations

An Overview of Null Features from C# 8+

Design intent

Design enforcement

Is the variable, parameter, field, return value, etc. supposed to allow null values?

Sometimes your intent is that a reference *can* represent nothing/no value (i.e. null).

Sometimes your intent is that a reference should *never* be nothing (i.e. always have a value).

	Ordinary Reference Type <C# 8	

	Ordinary Reference Type <C# 8	
Dereference		
Assign null		

	Ordinary Reference Type <C# 8	
Dereference	Yes (null check)	
Assign null		

	Ordinary Reference Type <C# 8	
Dereference	Yes (null check)	
Assign null	Yes	

	Ordinary Reference Type <C# 8	Non-Nullable Reference Type >= C# 8
Dereference	Yes (null check)	
Assign null	Yes	

	Ordinary Reference Type <C# 8	Non-Nullable Reference Type >= C# 8
Dereference	Yes (null check)	Yes
Assign null	Yes	

	Ordinary Reference Type <C# 8	Non-Nullable Reference Type >= C# 8
Dereference	Yes (null check)	Yes
Assign null	Yes	No

Enforced at compile time by compiler

Examine developer's design intent

Possible intent violations:

- Warning
- Error

Opt-in

Existing code

Nullable and Non-Nullable Generic Types

```
// Assume we are working in a nullable context

Message? nullMessage = null;
Message nonNullMessage = new();


List<Message> l1 = new();

l1.Add(nullMessage); // Warning: Possible null reference

l1.Add(nonNullMessage);


List<Message?> l2 = new();

l2.Add(nullMessage); // No warning

l2.Add(nonNullMessage);
```

```
// Assume we are working in a nullable context

class ObjectWriter<T>
{
    public void Write(T thingToWrite)
    {
        Console.WriteLine(thingToWrite);
    }
}

...

new ObjectWriter<int>() // non-nullable value type

new ObjectWriter<int?>() // nullable value type

new ObjectWriter<string>() // non-nullable reference type

new ObjectWriter<string?>() // nullable reference type
```

```
// Assume we are working in a nullable context

class ObjectWriterV2<T> where T : notnull
{
    public void Write(T thingToWrite)
    {
        Console.WriteLine(thingToWrite);
    }
}

...

new ObjectWriterV2<int>() // ok - non-nullable value type

new ObjectWriterV2<int?>() // Warning

new ObjectWriterV2<string>() // Ok - non-nullable ref type

new ObjectWriterV2<string?>() // Warning
```

```
// Assume we are working in a nullable context

class ObjectWriterV3<T> where T : class
{
    public void Write(T thingToWrite)
    {
        Console.WriteLine(thingToWrite);
    }
}

...

new ObjectWriterV3<int>(); // Error

new ObjectWriterV3<int?>(); // Error

new ObjectWriterV3<string>(); // Ok

new ObjectWriterV3<string?>(); // Warning
```

```
// Assume we are working in a nullable context

class ObjectWriterV4<T> where T : class?
{
    public void Write(T thingToWrite)
    {
        Console.WriteLine(thingToWrite);
    }
}

...

new ObjectWriterV4<int>(); // Error

new ObjectWriterV4<int?>(); // Error

new ObjectWriterV4<string>(); // Ok

new ObjectWriterV4<string?>(); // Ok
```

T?

Considerations

Don’t try to remove all null values from your code

Instead express your intent

Not completely null-safe

- Reflection
- Calling code that was compiled with feature disabled
- Analysis is limited in some cases*

Automated tests

```
// Assume we are working in a nullable context

public struct Email
{
    public string FromAddress; // non-nullable string
}


static void WriteEmail(Email email)
{
    Console.WriteLine(email.FromAddress.Length);
}


WriteEmail(default); // no warning but runtime exception
```

```
// Assume we are working in a nullable context

public struct DataContainer<T>
{
    public T Item1 { get; set; }
    public T Item2 { get; set; }
}


DataContainer<string> data = default;


// no warning but runtime exception
Console.WriteLine(data.Item1.Length);
```

```
// Assume we are working in a nullable context
string[] names = new string[10];
string firstName = names[0]; // no warning or exception

// no warning but runtime exception
Console.WriteLine(firstName.Length);
```

Summary

C# 8+ nulls: design intent & enforcement

#nullable enable & #nullable disable

<Nullable>enable</Nullable>

Treating nullable warnings as errors

? nullable operator, e.g. string?

Variables, properties, method returns

?? & ?.

Null-forgiving operator !

Refactored existing code

Generic constraints: notnull/class/class?

Considerations and limitations

Next:

Using Additional Attributes to Describe Nullability