Nullability in C#

JARED PARSONS

C# COMPILER LEAD

Introduction

- Member of C# Compiler and Language Design Team
- Working in C# production team for ~7 years
- Working in programming languages ~15 years

This Talk

- Design inspiration for nullable reference types
- Overview of the nullable feature set
  - Base nullable reference type feature set
  - C# 9 additions and changes
- Adoption strategies
- Nullability looking forward

Designing Nullable Reference Types

THINKING ABOUT NULLABILITY
IN THE LANDSCAPE OF C#

C# and null references

- C# 7.3 has no null reference tracking
  - Developers instead follow best practices
  - Read API documentation, guard APIs they're unsure of
  - Still get unexpected nulls in prod and testing
- C# language wants to fix this
  - How do you add nullable tracking to a 20 year old language?
  - And maintain compat while doing so?
  - And support a diverse ecosystem of libraries?

Creating vs. Extending

- Creating a language has unbounded options
  - Nothing to be compatible with
  - Change the type system, API focus, etc …
- Extending a language narrows the possible solutions
  - Existing features that never thought about nullability
  - Existing libraries that want safety but want to remain compatible
  - Must maintain the look and feel of the language

What about a perfect type solution?

- Perfectly separates nullable and non-nullable types
- C# variants exist with "perfect" null safety
  - Spec#, Sing#, System C#
  - Successful at implementing null safety
  - Unsuccessful at adoption because it simply wasn't C# anymore
- Breaks lots of existing patterns
  - Array creation like new string[4]
  - Calling methods in constructors
  - Using default(T) in generics

What about an API solution?

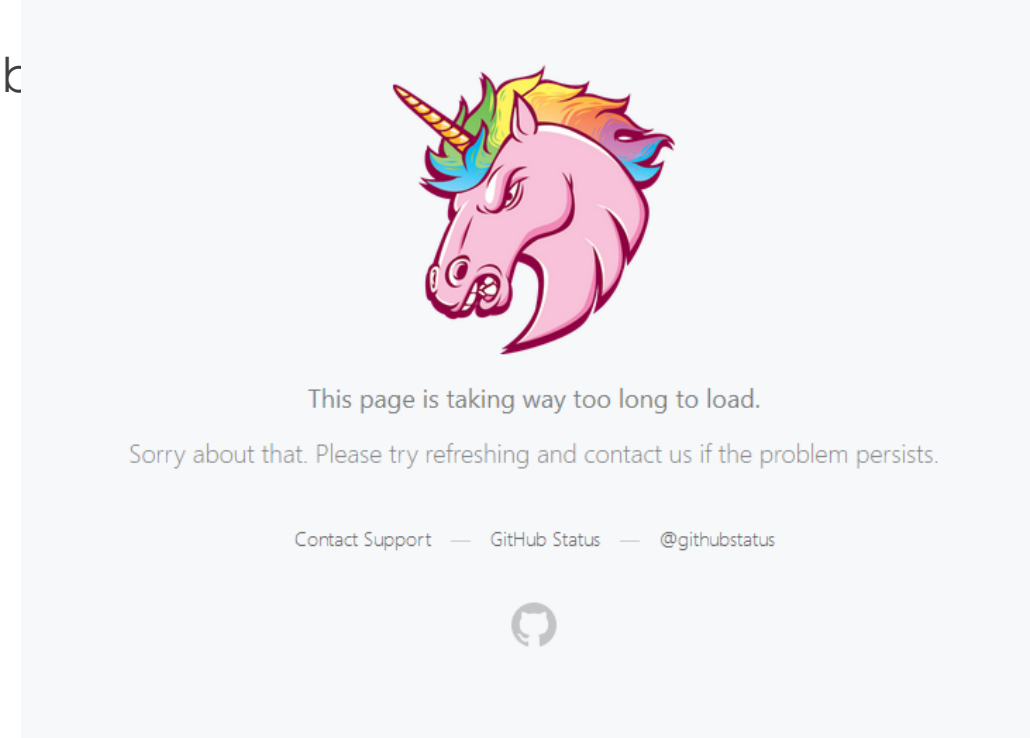
- Solution: Optional<T>
  - Nullable<T> but for reference types
  - Value cannot be used directly
  - Must do an explicit null check to get value
- Problem
  - Does nothing for existing code
  - Breaks binary compatibility
  - It's a yet another form of runtime null

Lots of C# code to annotate

- Code bases of varying size
  - Small: < 10,000 lines
  - Medium: < 100,000 lines
  - Large: < 1,000,000 lines
  - Jumbo: > 1,000,000 lines
- github/roslyn GitHub repository as an example has ~5,000,000 lines
- Null safety must be usable in all these code bases

Can’t enable in one change

- Cost of adopting nullability is proportional to code b
  - Medium code bases can change hundreds of files
  - Large code bases can change thousands
- 1,000 file pull requests are problematic
  - GitHub won’t display them well (if at all)
  - Reviewing is time consuming
- Must support incremental adoption
  - Annotate a single component, file or directory
  - Keep PRs managable and focused



.NET has a broad ecosystem

- Ecosystem
  - NuGet.org has ~175,000 unique packages
  - MyGet.org has thousands of customers
  - Private companies with proprietary libraries
- Developers want better null checking now
  - Can't wait for all dependencies to move first
  - Need to adopt independent of dependencies
- Developers will get nullable annotations in waves
  - Every NuGet update potentially gives new annotations
  - Cannot significantly increase cost of updating here

Principles for successful solution

- Embrace an imperfect solution
- Add value to existing code without major rewrite
- Provide nullable guidance through warnings
- Support incremental adoption

Non-Nullable Reference Types

AKA THE LAST THREE YEARS OF THE C# TEAM

Nullable is biggest feature since Generics

- C# 8
  - Language Design 9+ months
  - Compiler 2 devs for 1 year, majority of dev team for 7 months
  - .NET 5 annotated ~5,000 APIs
- C# 9
  - Language Design 3+ months (while doing records)
  - Compiler ~2 full time devs for the release
  - .NET 6 annotated ~11,000 APIs

Demo: Nullable Reference Types

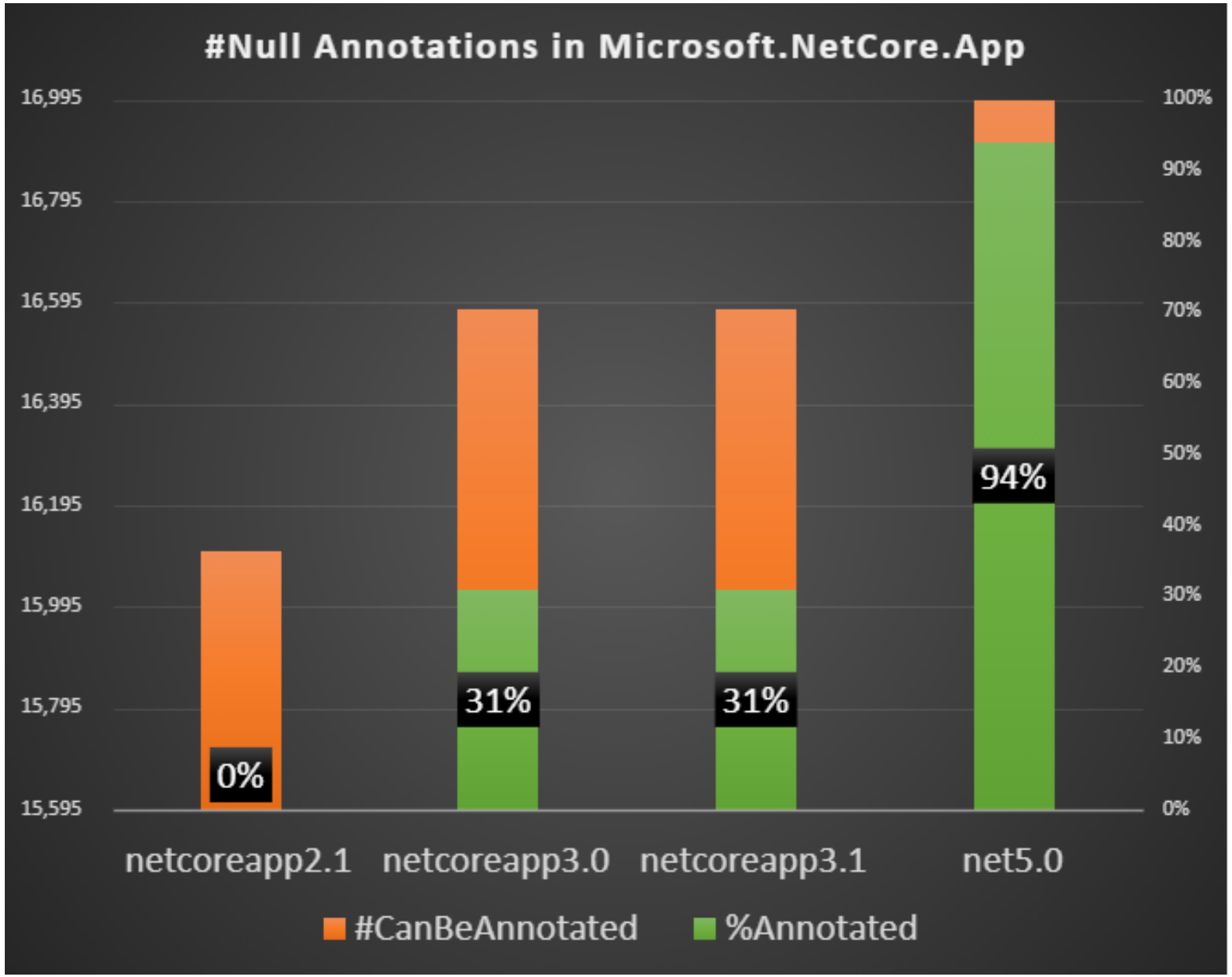
Demo: C# 9 Additions

Adoption Strategies

ADOPTING NULLABILITY IN YOUR CODE BASES

Which target framework to use?

Target Framework	Nullability Attributes Defined	Core Libraries Annotated
.NET Desktop	✗	✗
.NET Standard 2.0 (and before)	✗	✗
.NET Standard 2.1	✓	✗
.NET Core 2.1 (and before)	✗	✗
.NET Core 3.0 / 3.1	✓	Mscorlib Only
.NET Core 5.0	✓	✓


#Null Annotations in Microsoft.NetCore.App
16,995
16,795
16,595
16,395
16,195
15,995
15,795
15,595
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0%
31%
31%
94%
netcoreapp2.1
netcoreapp3.0
netcoreapp3.1
net5.0
#CanBeAnnotated
%Annotated

Libraries targeting older frameworks

- Libraries still target older target frameworks
  - netstandard2.0 for library breadth
  - netcoreapp3.0 / 3.1 because 5.0 is too new
- How can these libraries get the full API benefits of net5.0?
- Leverage multi-targeting
  - Multi-target net5.0 and netstandard2.0
  - Enable nullable warnings only on net5.0
  - Continue shipping netstandard2.0 only

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFrameworks>net5.0;netstandard2.0</TargetFrameworks>
  </PropertyGroup>

  <PropertyGroup Condition="'$(TargetFramework)' != 'net5.0'">
    <NoWarn>$(NoWarn);Nullable</NoWarn>
  </PropertyGroup>
</Project>
```

What order to annotate in?

- Annotate entire code base at once
- Annotate file when bug fixed
- Component by component
- Dependency Order
  - Start root project and move outwards
  - Annotate one project until done then move to next
- API to implementation
  - Annotate all public APIs first and implementations after
  - Annotating implementation should not impact other projects

Nullability Looking Forward

WHERE ARE WE GOING FROM HERE?

.NET Ecosystem

- .NET SDK
  - Core Libraries are annotated (~17,000+ APIs)
  - Roslyn, ASP.NET Core, EF Core, etc … in progress
- External Components
  - Increasing nullable use in the ecosystem
  - Examples: vs-threading, nUnit, xUnit, Netwonsoft.Json

C# Language

- C# 8 introduced nullable reference types
- C# 9 had substantial changes
  - All new features designed with nullability in mind
  - New language features
  - New attributes and enforcement
  - Tweaked the enforcement of rules to match customer scenarios
- C# 10+
  - Continue evolving nullable reference types to meet customer scenarios
  - These changes will be in the margins though
  - Focus is on helping reduce number of warnings

C# 10 intersection with nullability

- Direct impact
  - Additional attributes
  - More enforcement for C# 8 attributes
- Indirect impact
  - Extend definite assignment for “?.”, “??”, “==“ and “!=“.
  - Required properties
  - Parameterless struct constructors

Definite Assignment Change

```
if (map.TryGetValue(key, out var value) == true)
{
    Use(value);
}
Use(value);
```

```
if (map.TryGetValue(key, out var value) == true)
{
    Use(value);
}
// Error: `value` not definitely assigned
Use(value);
```

More Resources (1)

Blog posts:

- Try out Nullable Reference Types: devblogs.microsoft.com/dotnet/try-out-nullable-reference-types
- Nullable Reference Types in C#: devblogs.microsoft.com/dotnet/nullable-reference-types-in-csharp
- Nullable Reference Types Spec: https://github.com/dotnet/csharplang/blob/master/proposals/csharp-9.0/nullable-reference-types-specification.md

More Resources (2)

- NRTs overview: docs.microsoft.com/dotnet/csharp/nullable-references
- Nullable attributes: https://docs.microsoft.com/en-us/dotnet/csharp/nullable-attributes
- Update libraries to NRTs: docs.microsoft.com/dotnet/csharp/nullable-attributes
- NRTs tutorial: docs.microsoft.com/dotnet/csharp/tutorials/nullable-reference-types
- Migrate an app to NRTs: docs.microsoft.com/dotnet/csharp/tutorials/upgrade-to-nullable-references

Resources: Adoption Examples

- All at once
  - vs-threading 89 files
- Divide and Conquer
  - .NET SDK 3.0 annotations
  - .NET SDK 5.0 annotations
- Annotate as changed
  - C# compiler + IDE annotations

Q & A



- jaredpar@microsoft.com
- https://github.com/jaredpar
- https://twitter.com/jaredpar