DotNet

This document covers

## Overview

### Frameworks

A .NET Framework consists of

* Base Class Libraries
* Common Language Runtime

The choice of framework limits which platforms we can run our application on.

|  |  |
| --- | --- |
|  | Column Header |
| .NET Core | Open Source framework runs on Linux, MacOS and Windows |
| .NET Framework | Legacy Framework superceded by .NET Core |

The .NET Standard enables one to write common code that targets multiple frameworks

### C # Versions

#### 9.0

#### 8.0

* Nullable reference types
* Indices and Ranges
* Using declarations
* Null coalescing assignment
* Default interface members
* Switch Expression
* Tuple, positional and property patterns
* Asynchronous Streams

#### 7.0

* Tuples (System.ValueTuple)
* Ref local
* Ref return
* Ref Structs
* In modifier
* Stackalloc
* Out variables and discards
* Readonly struct – all fields enforced to be readonly
* Pattern variables
* Switch on types
* Local methods
* Expression bodied constructors
* Deconstructors

#### 6.0

* Null Conditional Operator (?.)
* Expression bodied functions
* String interpolation

Spans, Ref locals, stackalloc

## .Net Core

### Performance Enhancements

#### Tiered Compilation

Doing JIT compilation involves compromises. Using aggressive optimisations for every method leads to great steady state performance at the expense of longer start up time. Simpler compilation leads to faster start up at the cost of steady state throughout. ..NET framework did a single compilation that attempted to balance start-up costs and steady state performance.

Tiered compilation allows the same method to have multiple compilations that can be swapped at runtime. One compilation can be aimed at fast start up while another is aimed at steady state. At start-up, the JIT compiler generates a fast unoptimized compilation to facilitate fast start up. If the method is heavily used a background thread generates an optimised compilation that can be swapped in.

Most .NET core framework code loads from precompiled, ready to run images. These images lack some CPU optimisations. Where such methods are hot, they can also be recompiled at runtime for faster steady state performance.

On start-up time spent on JIT reduces by 35%. The amount of steady state performance probably depends how CPU bound the app is.

### JSON Serialisation (System.Text.Json)

Use Span and process UTF-8 directly without transcoding to UTF-16. For most tasks, the JSON serializer is 1.5 to 3 times faster. System.Text.JSON.

### Span<T>, Memory<T>

Span provides type-safe access to a contiguous area of memory. The memory can be located on the manager heap, the stack or even unmanaged memory. Span<T> is a ref struct which means it can never live on the manged heap. As such they can’t be boxed or assigned to variables of type object or interface types. They cannot be boxed or used as fields on classes or standard structs. The ref struct definition prevents any unnecessary heap allocations.

Span can be used to access substrings without allocation and copying.

string s = "John Smith";

// no allocation

System.ReadOnlySpan<char> span = "John Smith".AsSpan().Slice(5, 5);

// Allocation and copy

string sub = s.Substring(5, 5);

Internally a Span encapsulates a ref T that essentially is a direct pointer to some piece of memory. In this way it does not require an offset calculation to use it.