FullStack.Cafe - Tech Interview Plan

Q1: What is .NET Core? ☆

Topics: .NET Core

Answer:

The .NET Core platform is a new .NET stack that is optimized for open source development and agile delivery on NuGet.

.NET Core has two major components. It includes a small runtime that is built from the same codebase as the .NET Framework CLR. The .NET Core runtime includes the same GC and JIT (RyuJIT), but doesn't include features like Application Domains or Code Access Security. The runtime is delivered via NuGet, as part of the ASP.NET Core package.

.NET Core also includes the base class libraries. These libraries are largely the same code as the .NET Framework class libraries, but have been factored (removal of dependencies) to enable to ship a smaller set of libraries. These libraries are shipped as `System.*` NuGet packages on NuGet.org.

Q2: What is the difference between `String` and `string` in C#? ☆

Topics: .NET Core

Answer:

`string` is an *alias* in C# for `System.String`. So technically, there is no difference. It's like `int` vs. `System.Int32`.

As far as guidelines, it's generally recommended to use `string` any time you're referring to an object.

```
string place = "world";
```

Likewise, it's generally recommended to use `String` if you need to refer specifically to the class.

```
string greet = String.Format("Hello {0}!", place);
```

Q3: What is the .NET Framework? ☆

Topics: .NET Core

Answer:

The .NET is a Framework, which is a collection of classes of reusable libraries given by Microsoft to be used in other .NET applications and to develop, build and deploy many types of applications on the Windows platform including the following:

- Console Applications
- Windows Forms Applications
- Windows Presentation Foundation (WPF) Applications

- Web Applications
- Web Services
- Windows Services
- Services-oriented applications using Windows Communications Foundation (WCF)
- Workflow-enabled applications using Windows Workflow Foundation(WF)

Q4: What is .NET Standard? ☆

Topics: .NET Core

Answer:

The **.NET Standard** is a formal specification of .NET APIs that are intended to be available on all .NET implementations.

Q5: What are some characteristics of .NET Core? ☆☆

Topics: .NET Core

Answer:

- **Flexible deployment**: Can be included in your app or installed side-by-side user- or machine-wide.
- **Cross-platform**: Runs on Windows, macOS and Linux; can be ported to other OSes. The supported Operating Systems (OS), CPUs and application scenarios will grow over time, provided by Microsoft, other companies, and individuals.
- **Command-line tools**: All product scenarios can be exercised at the command-line.
- **Compatible**: .NET Core is compatible with .NET Framework, Xamarin and Mono, via the .NET Standard Library.
- **Open source**: The .NET Core platform is open source, using MIT and Apache 2 licenses. Documentation is licensed under CC-BY. .NET Core is a .NET Foundation project.
- **Supported by Microsoft**: .NET Core is supported by Microsoft, per .NET Core Support

Q6: What is the difference between .NET Core and Mono? ☆☆

Topics: .NET Core

Answer:

To be simple:

- Mono is third party implementation of .Net Framework for Linux/Android/iOs
- .Net Core is Microsoft's own implementation for same.

Q7: What's the difference between *SDK* and *Runtime* in .NET Core? ☆☆

Topics: .NET Core

Answer:

- The SDK is all of the stuff that is needed/makes developing a .NET Core application easier, such as the CLI and a compiler.
- The runtime is the "virtual machine" that hosts/runs the application and abstracts all the interaction with the base operating system.

Q8: What is the difference between `decimal`, `float` and `double` in .NET? ☆☆

Topics: .NET Core

Problem:

When would someone use one of these?

Solution:

Precision is the main difference.

- **Float** - 7 digits (32 bit)
- **Double** -15-16 digits (64 bit)
- **Decimal** -28-29 significant digits (128 bit)

As for what to use when:

- For values which are "naturally exact decimals" it's good to use **decimal**. This is usually suitable for any concepts invented by humans: financial values are the most obvious example, but there are others too. Consider the score given to divers or ice skaters, for example.
- For values which are more artefacts of nature which can't really be measured exactly anyway, **float/double** are more appropriate. For example, scientific data would usually be represented in this form. Here, the original values won't be "decimally accurate" to start with, so it's not important for the expected results to maintain the "decimal accuracy". Floating binary point types are much faster to work with than decimals.

Q9: What is an *unmanaged* resource in .NET? ☆☆

Topics: .NET Core

Answer:

Use that rule of thumb:

- If you found it in the Microsoft .NET Framework: it's **managed**.
- If you went poking around MSDN yourself, it's **unmanaged**.

Anything you've used P/Invoke calls to get outside of the nice comfy world of everything available to you in the .NET Framwork is unmanaged – and you're now *responsible* for cleaning it up.

Q10: What is *Boxing* and *Unboxing*? ☆☆

Topics: C# .NET Core

Answer:

Boxing and **Unboxing** both are used for type conversion but have some difference:

- **Boxing** - Boxing is the process of converting a value type data type to the object or to any interface data type which is implemented by this value type. When the CLR boxes a value means when CLR is converting a value type to Object Type, it wraps the value inside a `System.Object` and stores it on the heap area in application domain.

- **Unboxing** - Unboxing is also a process which is used to extract the value type from the object or any implemented interface type. Boxing may be done implicitly, but unboxing have to be explicit by code.

The concept of boxing and unboxing underlines the C# unified view of the type system in which a value of any type can be treated as an object.

Q11: What you understand by *Value types* and *Reference types* in .NET? Provide some comparison. ☆☆

Topics: C# .NET Core

Answer:

In C# data types can be of two types: **Value Types** and **Reference Types**. For a value type, the value is the information itself. For a reference type, the value is a reference which may be null or may be a way of navigating to an object containing the information.

- **Value types** - Holds some value not memory addresses. Example - Struct. A variable's value is stored wherever it is decleared. Local variables live on the stack for example, but when declared inside a class as a member it lives on the heap tightly coupled with the class it is declared in.

  Advantages: A value type does not need extra garbage collection. It gets garbage collected together with the instance it lives in. Local variables in methods get cleaned up upon method leave.

  Drawbacks:

  i. When large set of values are passed to a method the receiving variable actually copies so there are two redundant values in memory.
  ii. As classes are missed out.it losses all the oop benifits

- **Reference type** - Holds a memory address of a value not value. Example - Class. Stored on heap

  Advantages:

  i. When you pass a reference variable to a method and it changes it indeed changes the original value whereas in value types a copy of the given variable is taken and that's value is changed.
  ii. When the size of variable is bigger reference type is good
  iii. As classes come as a reference type variables, they give reusability, thus benefitting Object-oriented programming

  Drawbacks: More work referencing when allocating and dereferences when reading the value.extra overload for garbage collector

Q12: What is CLR? ☆☆

Topics: .NET Core

Answer:

The **CLR** stands for Common Language Runtime and it is an Execution Environment. It works as a layer between Operating Systems and the applications written in .NET languages that conforms to the Common Language Specification (CLS). The main function of Common Language Runtime (CLR) is to convert the Managed Code into native code and then execute the program.

Q13: Name some CLR services? ☆☆

Topics: .NET Core

Answer:

CLR services

- Assembly Resolver
- Assembly Loader
- Type Checker
- COM marshalled
- Debug Manager
- Thread Support
- IL to Native compiler
- Exception Manager
- Garbage Collector

Q14: What is CTS? ☆☆

Topics: .NET Core

Answer:

The **Common Type System (CTS)** standardizes the data types of all programming languages using .NET under the umbrella of .NET to a common data type for easy and smooth communication among these .NET languages.

CTS is designed as a singly rooted object hierarchy with `System.Object` as the base type from which all other types are derived. CTS supports two different kinds of types:

1. **Value Types**: Contain the values that need to be stored directly on the stack or allocated inline in a structure. They can be built-in (standard primitive types), user-defined (defined in source code) or enumerations (sets of enumerated values that are represented by labels but stored as a numeric type).
2. **Reference Types**: Store a reference to the value‘s memory address and are allocated on the heap. Reference types can be any of the pointer types, interface types or self-describing types (arrays and class types such as user-defined classes, boxed value types and delegates).

Q15: What is a .NET application domain? ☆☆

Topics: .NET Core

Answer:

It is an isolation layer provided by the .NET runtime. As such, App domains live with in a process (1 process can have many app domains) and have their own virtual address space.

App domains are useful because:

- They are less expensive than full processes
- They are multithreaded
- You can stop one without killing everything in the process
- Segregation of resources/config/etc
- Each app domain runs on its own security level

Q16: What is MSIL? ☆☆

Topics: .NET Core

Answer:

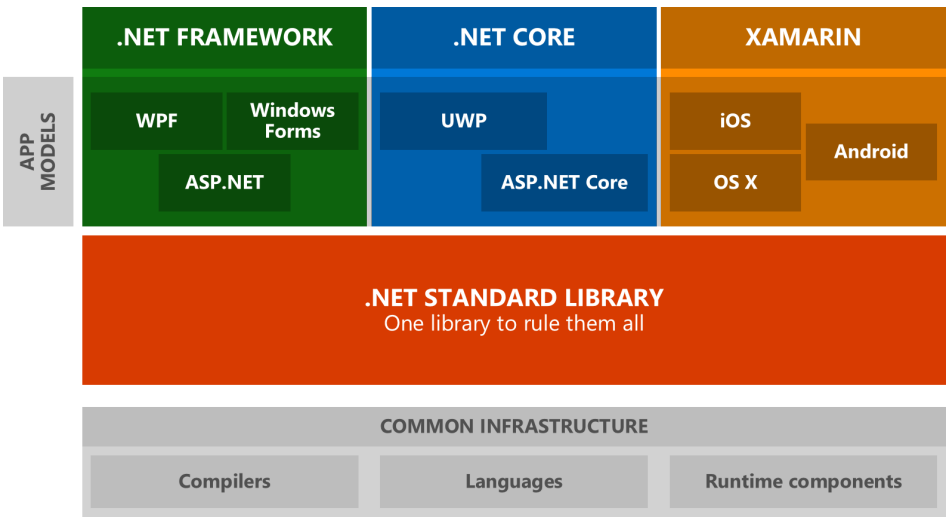
When we compile our .NET code then it is not directly converted to native/binary code; it is first converted into intermediate code known as MSIL code which is then interpreted by the CLR. MSIL is independent of hardware and the operating system. Cross language relationships are possible since MSIL is the same for all .NET languages. MSIL is further converted into native code.

Q17: What is .NET Standard and why we need to consider it? ☆☆

Topics: .NET Core

Answer:

1. **.NET Standard** solves the code sharing problem for .NET developers across all platforms by bringing all the APIs that you expect and love across the environments that you need: desktop applications, mobile apps & games, and cloud services:
2. **.NET Standard** is a **set of APIs** that **all** .NET platforms **have to implement**. This **unifies the .NET platforms** and **prevents future fragmentation**.
3. **.NET Standard 2.0** will be implemented by **.NET Framework**, .**NET Core**, and **Xamarin**. For **.NET Core**, this will add many of the existing APIs that have been requested.
4. **.NET Standard 2.0** includes a compatibility shim for **.NET Framework** binaries, significantly increasing the set of libraries that you can reference from your .NET Standard libraries.
5. **.NET Standard will replace Portable Class Libraries (PCLs)** as the tooling story for building multi-platform .NET libraries.



Q18: Explain what is included in .NET Core? ☆☆☆

Topics: .NET Core

Answer:

- A .NET runtime, which provides a type system, assembly loading, a garbage collector, native interop and other basic services.
- A set of framework libraries, which provide primitive data types, app composition types and fundamental utilities.
- A set of SDK tools and language compilers that enable the base developer experience, available in the .NET Core SDK.
- The 'dotnet' app host, which is used to launch .NET Core apps. It selects the runtime and hosts the runtime, provides an assembly loading policy and launches the app. The same host is also used to launch SDK tools in much the same way.

Q19: Explain two types of deployment for .NET Core applications ☆☆☆

Topics: .NET Core

Answer:

- **Framework-dependent deployment (FDD)** - it relies on the presence of a shared system-wide version of .NET Core on the target system. The app contains only its own code and any third-party dependencies that are outside of the .NET Core libraries. FDDs contain .dll files that can be launched by using the dotnet utility from the command line.

```
dotnet app.dll
```

- **Self-contained deployment** - Unlike FDD, a self-contained deployment (SCD) doesn't rely on the presence of shared components on the target system. All components, including both the .NET Core libraries and the .NET Core runtime, are included with the application and are isolated from other .NET Core applications. SCDs include an executable (such as app.exe on Windows platforms for an application named app), which is a renamed version of the platform-specific .NET Core host, and a .dll file (such as app.dll), which is the actual application.

Q20: Is there a way to catch *multiple* exceptions at once and without code duplication? ☆☆☆

Topics: .NET Core C#

Problem:

Consider:

```
try
{
    WebId = new Guid(queryString["web"]);
}
catch (FormatException)
{
    WebId = Guid.Empty;
}
catch (OverflowException)
{
```

```
    WebId = Guid.Empty;
}
```

Is there a way to catch both exceptions and only call the `WebId = Guid.Empty` call once?

Solution:

Catch `System.Exception` and switch on the types:

```
catch (Exception ex)
{
    if (ex is FormatException || ex is OverflowException)
    {
        WebId = Guid.Empty;
        return;
    }

    throw;
}
```