In C Sharp 3.0 Lambda expressions are introduced. So you can make use of lambda expressions instead of creating a function and then an instance of a delegate and then passing the function as a parameter to the delegate. The sample example rewritten using the Lambda expression is shown below. The private Promote method is no longer required now

**Filters in ASP.NET Core allow code to run before or after specific stages in the request processing pipeline**

* [OnAuthorization](https://docs.microsoft.com/en-us/previous-versions/aspnet/dd470545(v=vs.100))
* [OnException](https://docs.microsoft.com/en-us/previous-versions/aspnet/dd460532(v=vs.100))
* [OnActionExecuting](https://docs.microsoft.com/en-us/previous-versions/aspnet/dd493080(v=vs.100))
* [OnActionExecuted](https://docs.microsoft.com/en-us/previous-versions/aspnet/dd460283(v=vs.100))
* [OnResultExecuting](https://docs.microsoft.com/en-us/previous-versions/aspnet/dd504998(v=vs.100))
* [OnResultExecuted](https://docs.microsoft.com/en-us/previous-versions/aspnet/dd470615(v=vs.100))

Module is collection of components,servuces,directives,pipes and so on

Component ***The Component is a fundamental block of Angular and multiple components will make up your application.***

***Multiple component will combine and make your application***

Dependency inject is that like some functionality of services needed by various modules in application so by use of dependency injection mechanism we can inject these SERVICES /dependencies in our modules/components

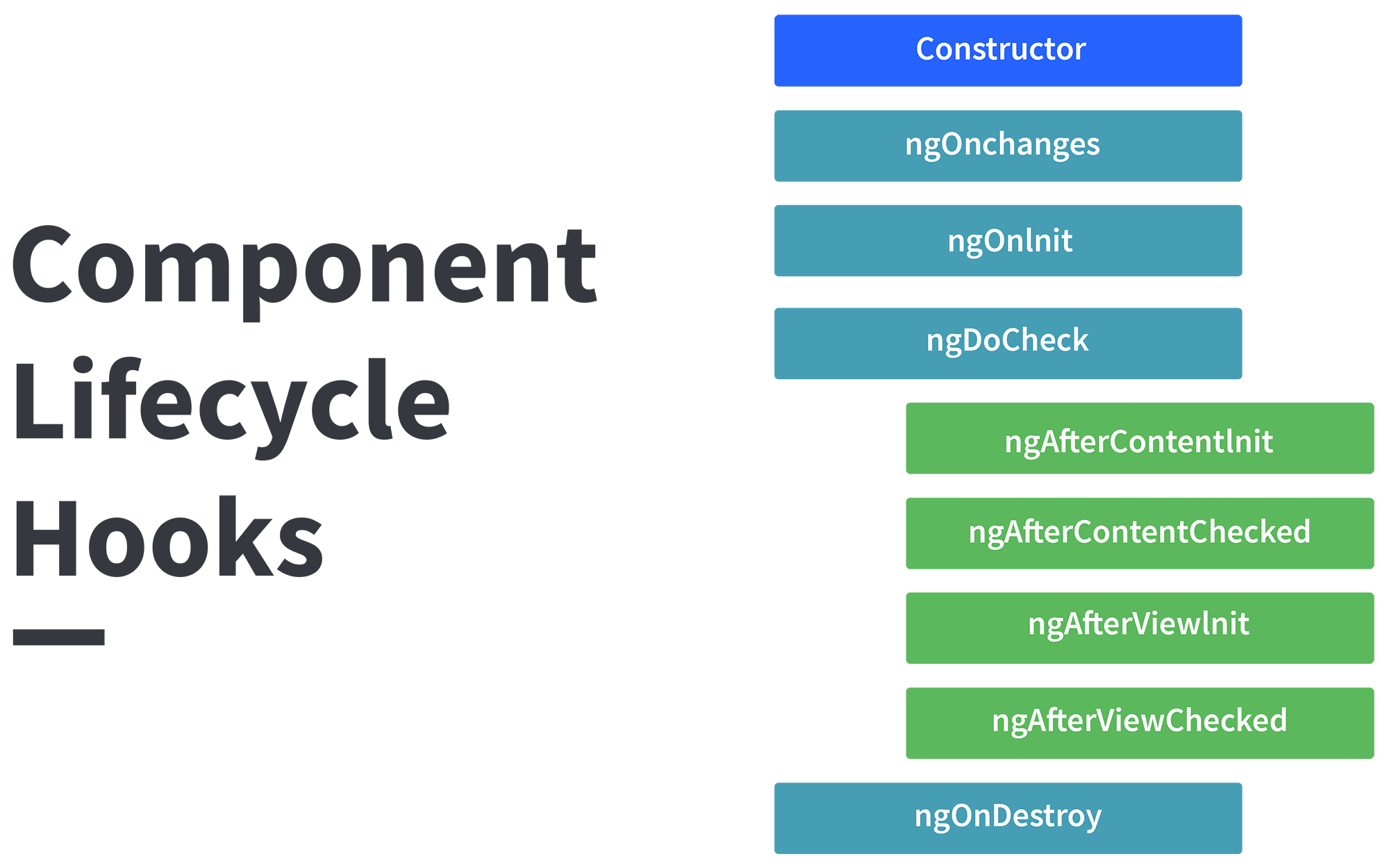
we are just making dependencies which are injectable across all MODULES of an application.

**Hashtable Vs Dictionary**

|  |  |
| --- | --- |
| Hashtable | Dictionary |
| A Hashtable is a non-generic collection. | A Dictionary is a generic collection. |
| Hashtable is defined under System.Collections namespace. | Dictionary is defined under System.Collections.Generic namespace. |
| In Hashtable, you can store key/value pairs of the same type or of the different type. | In Dictionary, you can store key/value pairs of same type. |
| In Hashtable, there is no need to specify the type of the key and value. | In Dictionary, you must specify the type of key and value. |
| The data retrieval is slower than Dictionary due to boxing/ unboxing. | The data retrieval is faster than Hashtable due to no boxing/ unboxing. |
| In Hashtable, if you try to access a key that doesn’t present in the given Hashtable, then it will give null values. | In Dictionary, if you try to access a key that doesn’t present in the given Dictionary, then it will give error. |
| It is thread safe. | It is also thread safe but only for public static members. |
| It doesn’t maintain the order of stored values. | It always maintain the order of stored values. |

### What are lifecycle hooks in Angular? Explain a few lifecycle hooks.

Every component in Angular has a lifecycle, different phases it goes through from the time of creation to the time it's destroyed. Angular provides **hooks** to tap into these phases and trigger changes at specific phases in a lifecycle.



**ngOnChanges( )** This hook/method is called before **ngOnInit** and whenever one or more input properties of the component changes.  
This method/hook receives a SimpleChanges object which contains the previous and current values of the property.  
  
**ngOnInit( )** This hook gets called once, after the **ngOnChanges** hook.  
It initializes the component and sets the input properties of the component.  
  
**ngDoCheck( )** It gets called after **ngOnChanges** and **ngOnInit** and is used to detect and act on changes that cannot be detected by Angular.  
We can implement our change detection algorithm in this hook. **ngAfterContentInit( )** It gets called after the first **ngDoCheck** hook. This hook responds after the content gets projected inside the component.  
  
**ngAfterContentChecked( )** It gets called after **ngAfterContentInit** and every subsequent **ngDoCheck**. It responds after the projected content is checked.  
  
**ngAfterViewInit( )** It responds after a component's view, or a child component's view is initialized.  
  
**ngAfterViewChecked( )** It gets called after **ngAfterViewInit**, and it responds after the component's view, or the child component's view is checked.  
  
**ngOnDestroy( )** It gets called just before Angular destroys the component. This hook can be used to clean up the code and detach event handlers.

### What are directives in Angular?

A directive is a class in Angular that is declared with a **@Directive** decorator.  
Every directive has its own behaviour and can be imported into various components of an application.  
  
**When to use a directive?**  
Consider an application, where multiple components need to have similar functionalities. The norm thing to do is by adding this functionality individually to every component but, this task is tedious to perform. In such a situation, one can create a **directive** having the required functionality and then, import the directive to components which require this functionality.  
  
**Types of directives**  
**Component directives**  
These form the main class in directives. **Instead** of @Directive decorator we use **@Component** decorator to declare these directives. These directives have a view, a stylesheet and a selector property.  
  
**Structural directives**  
These directives are generally used to manipulate DOM elements.  
Every structural directive has a ‘ \* ’ sign before them.  
We can apply these directives to any DOM element.

### 14. Explain the concept of Dependency Injection?

Dependency injection is an application design pattern which is implemented by Angular.  
It also forms one of the core concepts of Angular.  
  
**So what is dependency injection in simple terms?**  
Let’s break it down, dependencies in angular are nothing but services which have a functionality. Functionality of a service, can be needed by various components and directives in an application. Angular provides a smooth mechanism by which we can inject these dependencies in our components and directives.  
So basically, we are just making dependencies which are injectable across all components of an application.

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'angular';

}

**cloud computing is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the Internet (“the cloud”) to offer faster innovation, flexible resources, and economies of scale.**

Cloud Computing is the high-level abstraction procedure that focuses on business logic. This is a service delivered via the internet that aids you with the computing services without laying much importance on the infrastructural needs just like the electric supply.

#### What is meant by Microsoft Azure and Azure diagnostic?

**Answer:**This is one of the most basic Azure cloud interview questions asked very often. Microsoft **Azure is a cloud computing interface that is implemented by Microsoft so as to get benefited from cloud computing.**

Azure diagnostics is an API based system that collects the data to diagnose the application which is constantly running. It tunes with the verbose monitoring by enabling roles of the cloud services.

### . What is a method in C#?

**Sample answer:**

In C#, a method is a code block that contains a **series of statements** used to perform particular **operations**. Methods must be declared within a class or a structure. They help save time by reusing code.

### 8. What is meant by structure in C#?

**Sample answer:**

In C#, a structure is a **composite type** of data consisting of various **data types,** including methods, fields, constructors, constants, properties, indexers, operators, and even other structures.

**A structure helps bring various data types together under a single u**nit. In this way, **they are similar to classes. However, while classes are reference types, structures are value types.**

### 12. What is meant by garbage collection in C#?

**Sample answer:**

In C#, garbage collection is the **process of managing memory** in an application. The garbage collector automatically disposes of memory that is no longer used, to make memory available for new allocations.

### 13. What is a constructor in C#?\

constructor,:

Question: What are the constructors?

Answer: In C#, there is a special method that is invoked automatically at the time of object creation.

It initializes the data members of a new object and has the same name as the class or the structure. There are two types of constructors:

**Sample answer:**

In C#, a constructor is a special type of method that forms a part of a class. The main purpose of a **constructor is to initialize the fields of a class**. They are invoked automatically when a new class object is created.

### What is the difference between String and StringBuilder in C#?

**Sample answer:**

A string object is **immutable**, meaning that it cannot be changed after it’s created. Any operation that tries to modify the string object will simply create a new string object. On the other hand, a string builder object is mutable and can be modified as the developer sees fit.

### What is the difference between Const and ReadOnly keywords in C#?

**Sample answer:**

There are several differences between Const and ReadOnly keywords in C#. These include:

* ReadOnly is a constant used at runtime, whereas Const is a constant used at compile-time
* ReadOnly values can be changed, whereas Const values cannot be changed
* ReadOnly cannot be declared inside the method, whereas Const can

## **ViewData VS ViewBag VS TempData**

|  |  |  |
| --- | --- | --- |
| **ViewData** | **ViewBag** | **TempData** |
| It is Key-Value Dictionary collection | It is a type object | It is Key-Value Dictionary collection |
| ViewData is a dictionary object and it is property of ControllerBase class | ViewBag is Dynamic property of ControllerBase class. | TempData is a dictionary object and it is property of controllerBase class. |
| ViewData is Faster than ViewBag | ViewBag is slower than ViewData | NA |
| ViewData is introduced in MVC 1.0 and available in MVC 1.0 and above | ViewBag is introduced in MVC 3.0 and available in MVC 3.0 and above | TempData is also introduced in MVC1.0 and available in MVC 1.0 and above. |
| ViewData also works with .net framework 3.5 and above | ViewBag only works with .net framework 4.0 and above | TempData also works with .net framework 3.5 and above |
| Type Conversion code is required while enumerating | In depth, ViewBag is used dynamic, so there is no need to type conversion while enumerating. | Type Conversion code is required while enumerating |
| Its value becomes null if redirection has occurred. | Same as ViewData | TempData is used to pass data between two consecutive requests. |
| It lies only during the current request. | Same as ViewData | TempData only works during the current and subsequent request |

ViewBag is a dynamic object to pass the data from Controller to View

1. TempData is used to pass data from the current request to the next request
2. ViewData is used to pass data from controller to view.
3. It is derived from ViewDataDictionary class.
4. It is available for the current request only.
5. Requires typecasting for complex data types and checks for null values to avoid an error.
6. If redirection occurs, then its value becomes null.

|  |  |  |
| --- | --- | --- |
| ViewData is a dictionary object and it is property of ControllerBase class | ViewBag is Dynamic property of ControllerBase class. | TempData is a dictionary object and it is property of controllerBase class. |

s

**What is ViewState?**

ViewState is used to retain the state of server-side objects between page post backs.

**List the events in page life cycle.asp.net**

1) Page\_PreInit  
2) Page\_Init  
3) Page\_InitComplete  
4) Page\_PreLoad  
5) Page\_Load  
6) Page\_LoadComplete  
7) Page\_PreRender  
8) Render

Profiler

# SQL Server Profiler

* Article
* 04/21/2022
* 10 minutes to read
* 13 contributors

**Applies to:** yesSQL Server (all supported versions) YesAzure SQL Managed Instance

**SQL Server Profiler is an interface to create and manage traces and analyze and replay trace results**. Events are saved in a trace file that can later be analyzed or used to replay a specific series of steps when diagnosing a problem.

Solid principles of programing language

S:single responsibility principle

O:open closed

L:Liskov substistional principle

I:Interface segregation

D:dependency inversion

Design patterns

Four majar pillars in opps

Abstact classes and interfaces

Default specifier in interface(public)

Abstact class can use constructor

Why we use abstract classes() An abstract class cannot be instantiated. The purpose of an abstract class is **to provide a common definition of a base class that multiple derived classes can share**.

How can you stop class to be inheritance

If class is sealed is the any way to inhaerit if class is sealed

Ans :Extension method are use to

Design pattern singleton design pattern,

Runtime and complie time polymorphism

Deligates why we are using deligates

Delegates **allow methods to be passed as parameters**

**If function is void we want return type how can we achive**

**In,Ref and out keyword:**

**ref keyword is used when a called method has to update the passed parameter. out keyword is used when a called method has to update multiple parameter passed** . ref keyword is used to pass data in bi-directional way. out keyword is used to get data in uni-directional way.

Application life cycle of dot net core

Dot net core :every methos get complied there are some method exceuted request pilpelines, middlewares method

How many types of routing in asp.net core

What are the action methods and what are types of action methods

### Ans: What is ASP.NET Web API routing?

Routing is the most important part of ASP.NET Web API. Routing is a way how Web API matches a URI to an action. It is basically a process that decides which action and controller should be called.

The controller is basically a class that handles all HTTP requests. All public methods of controllers are basically known as action methods or just actions. Whenever a Web API framework receives any type of request, it routes that request to action.

There are basically two ways to implement routing in Web API as given below:  
**Convention-based routing**: Web API supports convention-based routing. In this type of routing, Web API uses route templates to select which controller and action method to execute.   
  
**Attribute-based routing**: Web API 2 generally supports a new type of routing known as attribute routing. As the name suggests, it uses attributes to define routes. It is the ability to add routes to the route table via attributes.

[GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET)

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

[HEAD](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD)

The HEAD method asks for a response identical to a GET request, but without the response body.

[POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST)

This verb should be used only to post or create new entry with information or data to database or other source. Code will look like the given below. The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.

[PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT)

This verb should be used only to update the existing entry with information or data to database or other source. The code will look like the given code.The PUT method replaces all current representations of the target resource with the request payload.

[DELETE](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE)

The DELETE method deletes the specified resource.

What kind of work do have the mvc

Web api

Can you just tell me what are http verb post, get,put,

http put and patch describe

## **view Vs Out**

## Ref and out keywords in C# are used to pass arguments within a method or function. Both indicate that an argument/parameter is passed by reference. By default parameters are passed to a method by value. By using these keywords (ref and out) we can pass a parameter by reference.

|  |  |
| --- | --- |
| **Ref** | **Out** |
| The parameter or argument must be initialized first before it is passed to ref. | It is not compulsory to initialize a parameter or argument before it is passed to an out. |
| It is not required to assign or initialize the value of a parameter (which is passed by ref) before returning to the calling method. | A called method is required to assign or initialize a value of a parameter (which is passed to an out) before returning to the calling method. |
| Passing a parameter value by Ref is useful when the called method is also needed to modify the pass parameter. | Declaring a parameter to an out method is useful when multiple values need to be returned from a function or method. |
| It is not compulsory to initialize a parameter value before using it in a calling method. | A parameter value must be initialized within the calling method before its use. |
| When we use REF, data can be passed bi-directionally. | When we use OUT data is passed only in a unidirectional way (from the called method to the caller method). |
| Both ref and out are treated differently at run time and they are treated the same at compile time. | |
| Properties are not variables, therefore it cannot be passed as an out or ref parameter. | |

**Cursor** is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on Table by User. Cursors are used to store Database Tables. There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained as following below.

1. **Implicit Cursors:**  
   Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
2. **Explicit Cursors :**  
   Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

sss

What are the action methods and what are types of action methods

# **Action method**

In this section, you will learn about the action method of the controller class.

All the public methods of the Controller class are called Action methods. They are like any other normal methods with the following restrictions:

1. Action method must be public. It cannot be private or protected
2. Action method cannot be overloaded
3. Action method cannot be a static method.

# **Controllers in ASP.NET MVC**

In this section, you will learn about the Controller in ASP.NET MVC.

The Controller in MVC architecture handles any incoming URL request. The Controller is a class, derived from the base class System.Web.Mvc.Controller. Controller class contains public methods called **Action** methods. Controller and its action method handles incoming browser requests, retrieves necessary model data and returns appropriate responses.

WHAT IS ACTION METHOD IN ASP.NET MVC 5?

All the public methods which are written inside a Controller are known as Action Method. When creating Action Method you must follow these rules.

a. Action method must be public.

b. It cannot be overloaded.

c. It cannot be a static method.

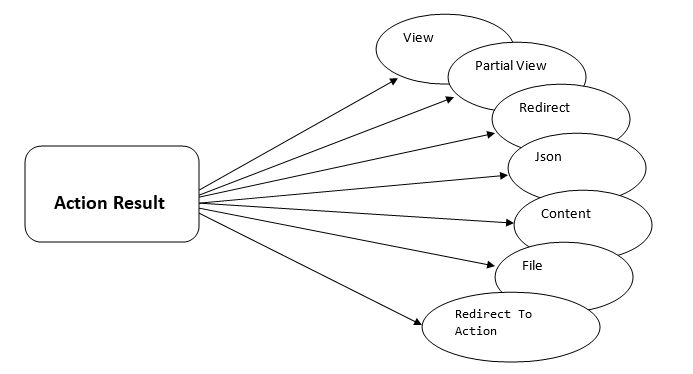
d. Every controller has at least one default Action method Index() that returns the view page.

e. ActionResult is a base class of all the result type action methods.

TYPES OF ACTION METHOD

Action Result

ActionResult is the base class of all the result type action method. There are following Result type action method in MVC.



ViewResult - Represents HTML and markup.

EmptyResult - Represents no result.

RedirectResult - Represents a redirection to a new URL.

JsonResult - Represents a JavaScript Object Notation result that can be used in an AJAX application.

JavaScriptResult - Represents a JavaScript script.

ContentResult - Represents a text result.

FileContentResult - Represents a downloadable file (with the binary content).

FilePathResult - Represents a downloadable file (with a path).

**Questions description:**Can we have virtual method in an Abstract class in C#? If yes, what is the purpose of having virtual method and abstract method both in an abstract class if both allow derived classes to override and implement it. for example class below.

abstract class Car

{

public virtual void speed() {Console.WriteLine("120 kmph");}

public abstract void mileage();

}

**Answer:**Yes, We can have virtual method in an Abstract class in C#.

This is true that both virtual and abstract method allow derived classes to override and implement it. But, difference is that **an abstract method forces derived classes to implement it** whereas **virtual method is optional**.

: **Yes, We can have virtual method in an Abstract class in C#**. This is true that both virtual and abstract method allow derived classes to override and implement it. But, difference is that an abstract method forces derived classes to implement it whereas virtual method is optional

**What is the difference between Server.Transfer and Response.Redirect?**

In Server.Transfer page processing transfers from one page to the other page without making a round-trip back to the client’s browser. This provides a faster response with a little less overhead on the server. The clients url history list or current url Server does not update in case of Server.Transfer.

Response.Redirect is used to redirect the user’s browser to another page or site. It performs trip back to the client where the client’s browser is redirected to the new page. The user’s browser history list is updated to reflect the new address.

## **Rules For Method Overriding**

* The [access modifier](https://www.edureka.co/blog/access-modifiers-in-java/) can only allow more access for the overridden method.
* A [final](https://www.edureka.co/blog/final-finally-and-finalize-in-java/) method does not support method overriding.
* A static method cannot be overridden.
* Private methods cannot be overridden.
* The return type of the overriding method must be the same.
* We can call the parent class method in the overriding method using the super keyword.
* A [constructor](https://www.edureka.co/blog/constructor-in-java/) cannot be overridden because a child class and a parent class cannot have the constructor with the same name.
* Method overloading and rrding
* When two or more methods in the same class have the same name but different parameters, it’s called Overloading.
* When the [method](https://www.journaldev.com/22385/java-method) signature (name and parameters) are the same in the superclass and the child class, it’s called Overriding.

Delegates:

* C# delegates are similar to pointers to functions, in C or C++. A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime.
* Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the **System.Delegate** class.

// C# program to illustrate the use of Delegates

using System;

namespace GeeksForGeeks {

// declare class "Geeks"

class Geeks {

// Declaring the delegates

// Here return type and parameter type should

// be same as the return type and parameter type

// of the two methods

// "addnum" and "subnum" are two delegate names

public delegate void addnum(int a, int b);

public delegate void subnum(int a, int b);

// method "sum"

public void sum(int a, int b)

{

Console.WriteLine("(100 + 40) = {0}", a + b);

}

// method "subtract"

public void subtract(int a, int b)

{

Console.WriteLine("(100 - 60) = {0}", a - b);

}

// Main Method

public static void Main(String []args)

{

// creating object "obj" of class "Geeks"

Geeks obj = new Geeks();

// creating object of delegate, name as "del\_obj1"

// for method "sum" and "del\_obj2" for method "subtract" &

// pass the parameter as the two methods by class object "obj"

// instantiating the delegates

addnum del\_obj1 = new addnum(obj.sum);

subnum del\_obj2 = new subnum(obj.subtract);

// pass the values to the methods by delegate object

del\_obj1(100, 40);

del\_obj2(100, 60);

// These can be written as using

// "Invoke" method

// del\_obj1.Invoke(100, 40);

// del\_obj2.Invoke(100, 60);

}

}

}

MVC Topics to learn:-

Tupple

How to take Multiple object  from controller to view

Tupple viewmodel

Expendoobject

SOLID Principle

# SOAP Vs. REST: Difference between Web API Services

[Alyssa Walker](https://www.guru99.com/author/alyssa)By[Alyssa Walker](https://www.guru99.com/author/alyssa)UpdatedApril 23, 2022

## What is SOAP?

**SOAP** is a protocol which was designed before REST and came into the picture. The main idea behind designing SOAP was to ensure that programs built on different platforms and programming languages could exchange data in an easy manner. SOAP stands for Simple Object Access Protocol.

## What is REST?

**REST** was designed specifically for working with components such as media components, files, or even objects on a particular hardware device. Any web service that is defined on the principles of REST can be called a RestFul web service. A Restful service would use the normal HTTP verbs of GET, POST, PUT and DELETE for working with the required components. REST stands for Representational State Transfer.

## KEY DIFFERENCE

* SOAP stands for Simple Object Access Protocol whereas REST stands for Representational State Transfer.
* SOAP is a protocol whereas REST is an architectural pattern.
* SOAP uses service interfaces to expose its functionality to client applications while REST uses Uniform Service locators to access to the components on the hardware device.
* SOAP needs more bandwidth for its usage whereas REST doesn’t need much bandwidth.
* Comparing SOAP vs REST API, SOAP only works with XML formats whereas REST work with plain text, XML, HTML and JSON.
* SOAP cannot make use of REST whereas REST can make use of SOAP.

## Difference Between SOAP and REST

Each technique has its own advantages and disadvantages. Hence, it’s always good to understand in which situations each design should be used. This REST and SOAP API difference tutorial will go into some of the key difference between REST and SOAP API as well as what challenges you might encounter while using them.Programming with Beginners Tutorial08:32Linux Tutorial for Beginners: Introduction to Linux Operating...01:35What is Integration Testing Software Testing Tutorial03:04What is JVM (Java Virtual Machine) with Architecture JAVA...02:24How to write a TEST CASE Software Testing Tutorial01:08Seven Testing Principles Software Testing05:01Linux File Permissions Commands with Examples13:29How to use Text tool in Photoshop CC Tutorial08:32What is NoSQL Database Tutorial02:00Important Linux Commands for Beginners Linux Tutorial15:03

Below is the main difference between SOAP and REST API

|  |  |
| --- | --- |
| **SOAP** | **REST** |
| * SOAP stands for Simple Object Access Protocol | * REST stands for Representational State Transfer |
| * SOAP is a protocol. SOAP was designed with a specification. It includes a WSDL file which has the required information on what the web service does in addition to the location of the web service. | * REST is an Architectural style in which a web service can only be treated as a RESTful service if it follows the constraints of being   1. Client Server   2. Stateless   3. Cacheable   4. Layered System   5. Uniform Interface |
| * SOAP cannot make use of REST since SOAP is a protocol and REST is an architectural pattern. | * REST can make use of SOAP as the underlying protocol for web services, because in the end it is just an architectural pattern. |
| * SOAP uses service interfaces to expose its functionality to client applications. In SOAP, the WSDL file provides the client with the necessary information which can be used to understand what services the web service can offer. | * REST use Uniform Service locators to access to the components on the hardware device. For example, if there is an object which represents the data of an employee hosted on a URL as http://demo.guru99 , the below are some of URI that can exist to access them.   http://demo.guru99.com/Employee  http://demo.guru99.com/Employee/1 |
| * SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot.   <?xml version="1.0"?>  <SOAP-ENV:Envelope  xmlns:SOAP-ENV  ="http://www.w3.org/2001/12/soap-envelope"  SOAP-ENV:encodingStyle  =" http://www.w3.org/2001/12/soap-encoding">  <soap:Body>  <Demo.guru99WebService  xmlns="http://tempuri.org/">  <EmployeeID>int</EmployeeID>  </Demo.guru99WebService>  </soap:Body>  </SOAP-ENV:Envelope> | * REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. Below is an example of a JSON message passed to a web server. You can see that the size of the message is comparatively smaller to SOAP.   {"city":"Mumbai","state":"Maharastra"} |
| * SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format. | * REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON. |

## When to use REST?

One of the most highly debatable topics is when REST should be used or when to use SOAP while designing web services. Below are some of the key factors that determine when REST and SOAP API technology should be used for web services **REST services should be used in the following instances**

* **Limited resources and bandwidth** – Since SOAP messages are heavier in content and consume a far greater bandwidth, REST should be used in instances where network bandwidth is a constraint.
* **Statelessness** – If there is no need to maintain a state of information from one request to another then REST should be used. If you need a proper information flow wherein some information from one request needs to flow into another then SOAP is more suited for that purpose. We can take the example of any online purchasing site. These sites normally need the user first to add items which need to be purchased to a cart. All of the cart items are then transferred to the payment page in order to complete the purchase. This is an example of an application which needs the state feature. The state of the cart items needs to be transferred to the payment page for further processing.
* **Caching**– If there is a need to cache a lot of requests then REST is the perfect solution. At times, clients could request for the same resource multiple times. This can increase the number of requests which are sent to the server. By implementing a cache, the most frequent queries results can be stored in an intermediate location. So whenever the client requests for a resource, it will first check the cache. If the resources exist then, it will not proceed to the server. So caching can help in minimizing the amount of trips which are made to the web server.
* **Ease of coding**– Coding REST Services and subsequent implementation is far easier than SOAP. So if a quick win solution is required for web services, then REST is the way to go.

Next in this SOAP and REST difference tutorial, we will learn when to use SOAP API.

## When to use SOAP?

SOAP should be used in the following instances

1. **Asynchronous processing and subsequent invocation** – if there is a requirement that the client needs a guaranteed level of reliability and security then the new SOAP standard of SOAP 1.2 provides a lot of additional features, especially when it comes to security.
2. **A Formal means of communication** – if both the client and server have an agreement on the exchange format then SOAP 1.2 gives the rigid specifications for this type of interaction. An example is an online purchasing site in which users add items to a cart before the payment is made. Let’s assume we have a web service that does the final payment. There can be a firm agreement that the web service will only accept the cart item name, unit price, and quantity. If such a scenario exists then, it’s always better to use the SOAP protocol.
3. **Stateful operations –**ifthe application has a requirement that state needs to be maintained from one request to another, then the SOAP 1.2 standard provides the WS\* structure to support such requirements.

Next in this REST vs SOAP API difference, we will learn about challenges with SOAP API.

## Challenges in SOAP API

API is known as the **Application Programming Interface** and is offered by both the client and the server. In the client world, this is offered by the browser whereas in the server world it’s what is provided by the web service which can either be SOAP or REST.

**Challenges with the SOAP API**

1. WSDL file – One of the key challenges of the SOAP API is the WSDL document itself. The WSDL document is what tells the client of all the operations that can be performed by the web service. The WSDL document will contain all information such as the data types being used in the SOAP messages and what all operations are available via the web service. The below code snippet is just part of a sample WSDL file.

<?xml version="1.0"?>

<definitions name="Tutorial"

targetNamespace=http://demo.guru99.com/Tutorial.wsdl

xmlns:tns=http://demo.guru99.com/Tutorial.wsdl

xmlns:xsd1=http://demo.guru99.com/Tutorial.xsd

xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/

xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>

<schema targetNamespace=http://Demo.guru99.com/Tutorial.xsd

xmlns="http://www.w3.org/2000/10/XMLSchema">

<element name="TutorialNameRequest">

<complexType>

<all>

<element name="TutorialName" type="string"/>

</all>

</complexType>

</element>

<element name="TutorialIDRequest">

<complexType>

<all>

<element name="TutorialID" type="number"/>

</all>

</complexType>

</element>

</schema>

</types>

As per the above WSDL file, we have an element called “TutorialName” which is of the type String which is part of the element TutorialNameRequest.

Now, suppose if the WSDL file were to change as per the business requirements and the TutorialName has to become TutorialDescription. This would mean that all the clients who are currently connecting to this web service would then need to make this corresponding change in their code to accommodate the change in the WSDL file.

This shows the biggest challenge of the WSDL file which is the tight contract between the client and the server and that one change could cause a large impact, on the whole, client applications.

1. Document size – The other key challenge is the size of the SOAP messages which get transferred from the client to the server. Because of the large messages, using SOAP in places where bandwidth is a constraint can be a big issue.

Next in this RESTful vs SOAP difference, we will learn about challenges with REST API.

## Challenges in REST API

1. **Lack of Security**– REST does not impose any sort of security like SOAP. This is why REST is very appropriate for public available URL’s, but when it comes down to confidential data being passed between the client and the server, REST is the worst mechanism to be used for web services.
2. **Lack of state**– Most web applications require a stateful mechanism. For example, if you had a purchasing site which had the mechanism of having a shopping cart, it is required to know the number of items in the shopping cart before the actual purchase is made. Unfortunately, the burden of maintaining this state lies with the client, which just makes the client application heavier and difficult to maintain.

## Difference between SOAP Vs CORBA Vs DCOM Vs Java RMI

Remote access techniques such as the RPC ([Remote Procedure calls](https://www.guru99.com/remote-procedure-call-rpc.html)) methods were in common use before SOAP and REST API came along. The various remote access techniques which were available are mentioned below.

1. **CORBA**– This was known as **C**ommon **O**bject **R**equest **B**roker **A**rchitecture. This system was put in place to ensure that applications built on various platforms could talk to each other. CORBA was based on an object-oriented architecture, but it was not necessary for the calling application to be based on this architecture. The major disadvantage of this technique was that it has to be developed in a separate language called the Interface Definition Language, and it just presented an additional language that had to be learned by developers to make use of the CORBA system.
2. **DCOM** – This is the **D**istributed **C**omponent **O**bject **M**odel, which is a proprietary Microsoft technology for clients to access remote components. The biggest issue with this mechanism was it was up to the client application to free up resources when no longer required.Secondly, when the client sent the request, it was up to the client to ensure that the request was wrapped or marshaled in a correct way so that the web service could understand the request sent. Another issue was if the client application was a[Java](https://www.guru99.com/java-tutorial.html)based application which had to work DCOM (Microsoft Technology) additional coding was required to ensure that applications built in other programming languages could work with DCOM based web services.
3. **Java RMI** – Known as Java **R**emote **M**ethod **I**nvocation, this was Java implementation on how remote objects could be called through remote procedure calls. The biggest restriction of this technology was that Java RMI could only be run on a Java Virtual Machine. This meant that the calling application also has to be run on the Java framework in order to make use of Java RMI.

The main differences between SOAP and these techniques are as follows

1. **Working over HTTP** – All of the RPC techniques have one big limitation, and it is that they don’t work by the HTTP protocol. Since all applications on the web had to work on this protocol, this used to be a major roadblock for clients which had to access these RPC-style web services.
2. **Working with non-standard ports** – Since the RPC style web services did not work by the HTTP protocol, separate ports had to be open for them for clients to access the functionality from these web services.

Where you have used delegates in your project

Greatest achivmnts

Javascript vs java

Code output

Partial class

Temp class

##### **Where did you use delegates in your project? Or how did you use delegates in your project?**

The Delegate is one of the very important aspects to understand. Most of the interviewers ask you to explain the usage of delegates in a real-time project that you have worked on. Delegates are extensively used by framework developers. Let us say we have a class called Employee as shown below.

###### **Employee Class**

**public** **class** Employee

**{**

**public** **int** ID **{** **get**; **set**; **}**

**public** **string** Name **{** **get**; **set**; **}**

**public** **int** Experience **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**}**

The **Employee** class has the following properties.

1. **Id**
2. **Name**
3. **Experience**
4. **Salary**

Now I want to write a method in the Employee class which can be used to promote employees. The method should take a list of Employee objects as a parameter and should print the names of all the employees who are eligible for a promotion. But the logic based on which the employee gets promoted should not be hardcoded. At times we may promote employees based on their experience and at times we may promote them based on their salary or maybe some other condition. So, the logic to promote employees should not be hard-coded within the method.

##### **How to achieve?**

To achieve this we can make use of delegates. So now I would design my class as shown below. We also created a delegate EligibleToPromotion. This delegate takes the Employee object as a parameter and returns a boolean. In the Employee class, we have the PromoteEmpoloyee method. This method takes a list of Employees and a Delegate of the type EligibleToPromotion as parameters. The method then loops through each employee object and passes it to the delegate. If the delegate returns true, then the Employee is promoted, else not promoted. So within the method, we have not hardcoded any logic on how we want to promote employees.

**namespace** *DelegateDemo*

**{**

**public** **delegate** **bool** EligibleToPromotion**(**Employee EmployeeToPromotion**)**;

**public** **class** Employee

**{**

**public** **int** ID **{** **get**; **set**; **}**

**public** **string** Name **{** **get**; **set**; **}**

**public** **int** Experience **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** **static** **void** PromoteEmployee**(**List**<**Employee**>** lstEmployees, EligibleToPromotion IsEmployeeEligible**)**

**{**

**foreach** **(**Employee employee in lstEmployees**)**

**{**

**if** **(**IsEmployeeEligible**(**employee**))**

**{**

Console.WriteLine**(**"Employee {0} Promoted", employee.Name**)**;

**}**

**}**

**}**

**}**

**}**

So now the client who uses the Employee class has the flexibility of determining the logic on how they want to promote their employees as shown below. First create the employee objects – E1, E2, and E3. Populate the properties for the respective objects. We then create an employeeList to hold all the 3 employees.

Notice the Promote method that we have created. This method has the logic of how we want to promote our employees. The method is then passed as a parameter to the delegate. Also, note this method has the same signature as that of the EligibleToPromotion delegate. This is very important because the Promote method cannot be passed as a parameter to the delegate if the signature differs. This is the reason why delegates are called type-safe function pointers.

**namespace** *DelegateDemo*

**{**

**public** **class** Employee

**{**

**public** **int** ID **{** **get**; **set**; **}**

**public** **string** Name **{** **get**; **set**; **}**

**public** **int** Experience **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** **static** **void** PromoteEmployee**(**List**<**Employee**>** lstEmployees, EligibleToPromotion IsEmployeeEligible**)**

**{**

**foreach** **(**Employee employee in lstEmployees**)**

**{**

**if** **(**IsEmployeeEligible**(**employee**))**

**{**

Console.WriteLine**(**"Employee {0} Promoted", employee.Name**)**;

**}**

**}**

**}**

**}**

**class** Program

**{**

**static** **void** Main**()**

**{**

Employee emp1 = new Employee**()**

**{**

ID = 101,

Name = "Pranaya",

Experience = 5,

Salary = 10000

**}**;

Employee emp2 = new Employee**()**

**{**

ID = 102,

Name = "Kumar",

Experience = 10,

Salary = 20000

**}**;

Employee emp3 = new Employee**()**

**{**

ID = 103,

Name = "Rout",

Experience = 20,

Salary = 30000

**}**;

List**<**Employee**>** lstEmployess = new List**<**Employee**>()**;

lstEmployess.Add**(**emp1**)**;

lstEmployess.Add**(**emp2**)**;

lstEmployess.Add**(**emp3**)**;

EligibleToPromotion eligibleTopromote = new EligibleToPromotion**(**Program.Promote**)**;

Employee.PromoteEmployee**(**lstEmployess, eligibleTopromote**)**;

Console.ReadKey**()**;

**}**

**public** **static** **bool** Promote**(**Employee employee**)**

**{**

**if** **(**employee.Salary **>** 10000**)**

**{**

**return** **true**;

**}**

**else**

**{**

**return** **false**;

**}**

**}**

**}**

**}**

So if we did not have the concept of delegates it would not have been possible to pass a function as a parameter. As the Promote method in the Employee class makes use of delegate, it is possible to dynamically decide the logic on how we want to promote employees.

##### **Using Lambda expressions**

In C Sharp 3.0 Lambda expressions are introduced. So you can make use of lambda expressions instead of creating a function and then an instance of a delegate and then passing the function as a parameter to the delegate. The sample example rewritten using the Lambda expression is shown below. The private Promote method is no longer required now.

**class** Program

**{**

**static** **void** Main**()**

**{**

Employee emp1 = new Employee**()**

**{**

ID = 101,

Name = "Pranaya",

Experience = 5,

Salary = 10000

**}**;

Employee emp2 = new Employee**()**

**{**

ID = 102,

Name = "Kumar",

Experience = 10,

Salary = 20000

**}**;

Employee emp3 = new Employee**()**

**{**

ID = 103,

Name = "Rout",

Experience = 20,

Salary = 30000

**}**;

List**<**Employee**>** lstEmployess = new List**<**Employee**>()**;

lstEmployess.Add**(**emp1**)**;

lstEmployess.Add**(**emp2**)**;

lstEmployess.Add**(**emp3**)**;

Employee.PromoteEmployee**(**lstEmployess, x =**>** x.Experience **>** 5**)**;

**}**

**}**