## JavaScript

JavaScript evaluates expressions from left to right.

JavaScript Types are Dynamic. JavaScript has dynamic types. This means that the same variable can be used to hold different data types

**JavaScript Arrays:**

JavaScript arrays are written with square brackets.

Array items are separated by commas.

var cars = ["Saab", "Volvo", "BMW"];

JavaScript ‘**typeof’ operator** to find the type of a JavaScript variable.

**Difference Between Undefined and Null:**

undefined and null are equal in value but different in type:

typeof undefined           // undefined  
typeof null                // object  
  
null === undefined         // false  
null == undefined          // true

**JavaScript Objects:**

JavaScript objects are written with curly braces {}.

Object properties are written as name:value pairs, separated by commas.

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

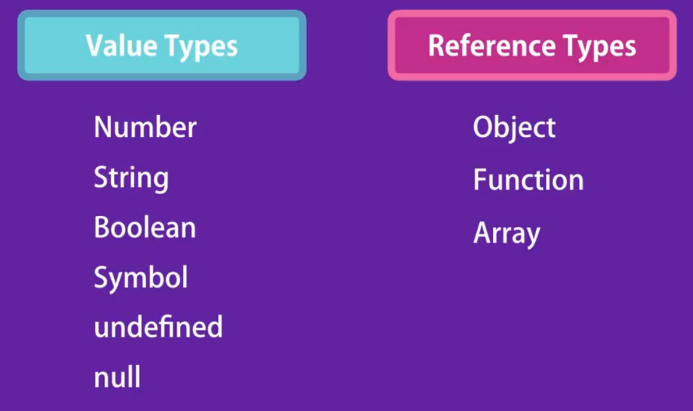
Property Property Value

firstName John

### JavaScript Data Types

Value Types are also called primitive type

Reference Types are also called objects types



Primitive Types (Value Types) are copied by their value.



Both x and y points to different location.

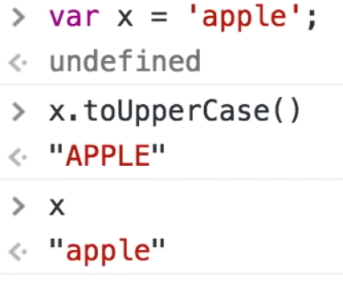
Object Types (Reference Types) are copied by their reference.



Both x and y are referring the same memory address. That’s why when we change the value of x and value of y is change.

Value Type are immutable (No changeable in the same object. Every time it creates a different object).

Value type examples:

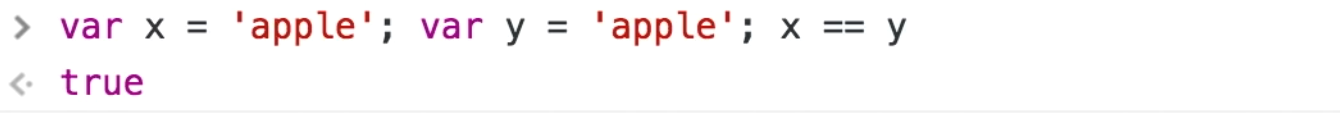


Reference Type are mutable (Changeable in the same object).

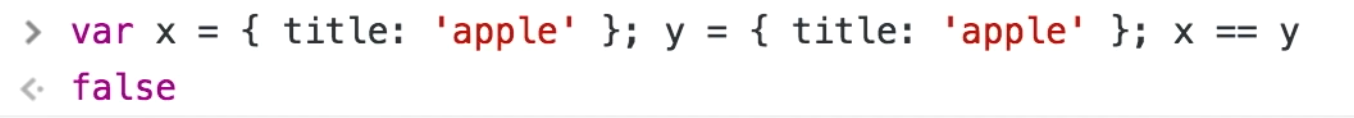
Reference Type Examples:

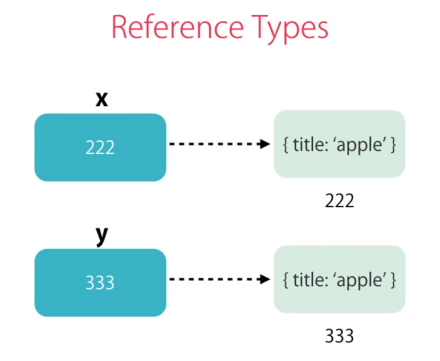


Value types are **compared by value**.



Reference type:





Value type are **copied by value.**

## AngularJS

GET:

var response = $http({

method: 'GET',

url: '/api/V1/Getmethod?Id=' + Id,

headers: { "Accept": "application/json;odata=verbose" }

});

POST:

var response = $http.post("/api/V1/SaveMethod",

JSON.stringify(data)

);

### AngularJS Custom Directives

**Angular JS Custom Directives:**

var myApp = angular.module('docsTemplateUrlDirective', []);

myApp.controller('myController', ['$scope', function($scope) {

$scope.customer = {

name: 'Pratik',

address: 'Indore'

};

}]);

myApp.directive('myCustomer', function() {

return {

templateUrl: 'my-customer.html'

};

});

HTML:

<div ng-controller="Controller">

<div my-customer></div>

</div>

my-customer.html

Name: {{customer.name}} Address: {{customer.address}}

### AngularJS Component

**Angular JS Component:**

In AngularJS, a Component is a special kind of directive that uses a simpler configuration which is suitable for a component-based application structure.

Advantages of Components:

1. simpler configuration than plain directives
2. optimized for component-based architecture
3. writing component directives will make it easier to upgrade to Angular

When **not to use** Components: when you want a directive that is triggered by an attribute or CSS class, rather than an element

**Creating and configuring a Component**

Components can be registered using the .component() method of an AngularJS module (returned by angular.module()). The method takes two arguments:

The name of the Component (as string).

The Component config object. (Note that, unlike the .directive() method, this method does not take a factory function.)

<https://www.linkedin.com/pulse/angularjs-1x-directives-vs-components-sachila-ranawaka>

AngularJS Component Vs Directive:

1. **Components are not a replacement for directives.** Actually component itself is a special kind of directive. For one thing, a component is always elements ('E') where directives can be an element, attribute, class or comment ('E','A','C','M'). A Component is an element then it should always require a template right? YES, a template is a mandatory property to a component whereas template is not mandatory in directives.
2. **Component-based structure:**

projectApp.component("ComponentName", {

controller: ControllerName,

controllerAs: "vm",

templateUrl: “CompnentName.view.html'

});

<div ng-app="projectApp">

< component-name></ component-name >

</div>

## SQL Questions and Answers

### Find nth highest salary in different ways

Find highest salary of each department in sql server:

--1.Using Inline query

SELECT e.EmpName, e.[Salary], d.DepName

FROM [Emp].[dbo].[tblEmployee] e

inner join [Emp].[dbo].[tblDepartment] d on e.DepId=d.DepId

where e.Salary in (SELECT MAX(Salary) FROM [Emp].[dbo].[tblEmployee] GROUP BY DepId)

--======================================================================================

--2.using CTE

;with cte\_HighestSalary(DepId,Salary) AS

(

SELECT DepId, MAX(Salary)as Salary FROM [Emp].[dbo].[tblEmployee] GROUP BY DepId

)

SELECT e.EmpName, e.Salary, e.DepId

FROM [Emp].[dbo].[tblEmployee] e

inner join cte\_HighestSalary d on e.Salary=d.Salary

order by DepId

--Nth highest salary of employee

select top 1 salary,EmpName from

(

select top N salary,EmpName

FROM [Emp].[dbo].[tblEmployee]

order by Salary desc

)temp

order by Salary asc

### Types of triggers

<https://docs.microsoft.com/en-us/sql/relational-databases/triggers/dml-triggers?view=sql-server-2017>

<https://www.dotnettricks.com/learn/sqlserver/after-trigger-instead-of-trigger-example>

| **Function** | **AFTER trigger** | **INSTEAD OF trigger** |
| --- | --- | --- |
| Applicability | Tables | Tables and views |
| Quantity per table or view | Multiple per triggering action (UPDATE, DELETE, and INSERT) | One per triggering action (UPDATE, DELETE, and INSERT) |
| Cascading references | No restrictions apply | INSTEAD OF UPDATE and DELETE triggers are not allowed on tables that are targets of cascaded referential integrity constraints. |
| Execution | After:  Constraint processing  Declarative referential actions  **inserted** and **deleted** tables creation  The triggering action | Before: Constraint processing  In place of: The triggering action  After: **inserted** and **deleted** tables creation |
| Order of execution | First and last execution may be specified | Not applicable |
| **varchar(max)**, **nvarchar(max)**, and **varbinary(max)** column references in **inserted** and **deleted** tables | Allowed | Allowed |
| **text**, **ntext**, and **image** column references in **inserted** and **deleted** tables | Not allowed | Allowed |

### SQL Server Constraints

Constraints in SQL Server are rules and restrictions applied on a column or a table such that unwanted data can't be inserted into tables. This ensures the accuracy and reliability of the data in the database. We can create constraints on single or multiple columns of any table. Constraints maintain the data integrity and accuracy in the table.

### Types of Constraints

### Column Type Constraints

Definitions of these types of constraints is given when the table is created.

1. **Create** **Table** My\_Constraint
2. (
3. IID **int** NOT NULL,
4. Salary **int** **CHECK**(Salary>5000)
5. )

### Table Type Constraints

Definitions of these types of constraints is given after the creation of the table using the Alter Command.

1. **Alter** **Table** My\_Cosntraint
2. **Add** **constraint** Check\_Constraint **Check**(Age>50)

SQL Server contains the following 6 types of constraints:

* Not Null Constraint
* Check Constraint
* Default Constraint
* Unique Constraint
* Primary Constraint
* Foreign Constraint

**Static in c#:**

The static keyword can be applied to a class members including constructors, methods, properties, and events. Static can also be applied to classes.

The static modifier makes an a member non-instantiable. The members of the class can be accessed via the class name (type) only.

**Static members in C# - When are they loaded and how is the memory allocation**

A class is loaded when either

* The static members of the class is accessed for the first time, or
* When the first instance of the class is created.

A class once loaded will remain in memory permanently and also are all static members of that class.

Series of events that occur when the first object is created,

1. Class is loaded.
2. **Static members are loaded and allocated memory.**
3. **Static constructor is executed.**
4. Object is created and Instance members are loaded and allocated memory.
5. Instance constructor is executed.
6. 4 and 5 repeat for subsequent object.

**Static Class**

A class in C# can be declared as static and such a class can have only static members and instance members are not allowed.

Static members in non-static class

The instance variable can only access the non-static methods and variables, it cannot access the static methods and variables.

public class MyNonStaticClass

{

private static int myStaticVariable = 0;

public static void MyStaticMethod()

{

Console.WriteLine("This is static method.");

}

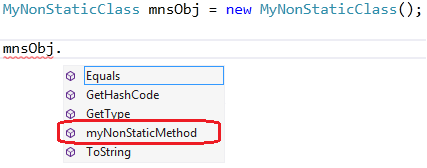
public void myNonStaticMethod()

{

Console.WriteLine("Non-static method");

}

}



**Static class in multiuser environment:**

Does static variables retain their values across user sessions?

Yes, Static fields hold their previous values.

Stack vs Heap:

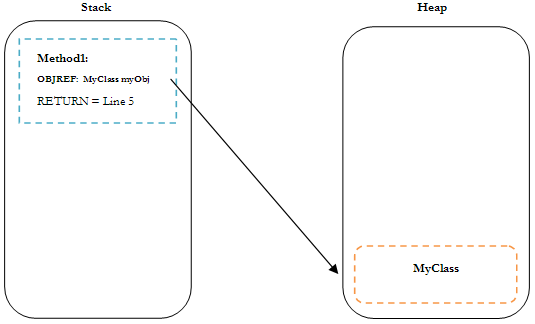
When an instance of a reference type is created (usually involving the new keyword), only an *object reference* is stored on stack. The actual instance itself is created on the heap, and its’ address held on the stack.

Consider the following code:

|  |  |
| --- | --- |
|  | 1 void Method1()  2 {  3      MyClass myObj=new MyClass();  4      Console.WriteLine(myObj.Text);  5 } |

**Listing 2: Code example using a reference type**

In Listing 2**.** a new instance of the class MyClass is created within the Method1 call.

****

Constructors:

[1]

public class Class1

{

static Class1()

{

Console.Write("Static");

}

**public Class1()**

**{**

**Console.Write("Default");**

**}**

**private Class1() // Because Default is already exist without params**

**{**

**Console.Write("Private");**

**}**

public Class1(int a, int b)

{

Console.Write("Param: "+ a.ToString()+"," +b.ToString());

}

}

When we create object of Class1

Class1 obj=new Class1(); **//Error**

Then it will give error: Type 'Class1' already defines a member called '.ctor' with the same parameter types

[2]

public class Class1

{

static Class1()

{

Console.Write("Static");

}

**public Class1()**

**{**

**Console.Write("Default");**

**}**

**private Class1(int p) {**

**Console.Write("Private");**

**}**

public Class1(int a, int b)

{

Console.Write("Param: "+ a.ToString()+"," +b.ToString());

}

}

When we create object of Class1

Class1 obj=new Class1(); **//Without any parameter will work**

Class1 obj=new Class1(3); **//Error because we cannot create instance of class that contains private constructor.**

Then it will give error: 'Class1.Class1(int)' is inaccessible due to its protection level

1. GAC? What is advantage of GAC? Diff between GAC and root bin dll
2. What is strong assembly
3. CTE vs Temp Table
4. What is referential integrity constraints?
5. What is DataReader ? Parameters of datareader?
6. Private constructor? What is use of it?
7. How can we access class members without create instance in private contructor class?
8. Use of private contructor in Singleton?
9. Diff between var and dynamic? Boxing and Unboxing
10. Diff between Dictionary and Hashtable? Can we store Key and values in different data types in hashtable?
11. Why we called string belongs to reference type?
12. How you estimate the requirement?
13. Write a query to get the output below

|  |  |  |
| --- | --- | --- |
| EmpId | EmpName | JoiningDate |
| 1 | Pratik | 2015-04-29 |
| 2 | Pankaj | 2015-05-15 |
| 3 | Anurag | 2014-01-01 |

Output:

|  |  |
| --- | --- |
| No. of Emp | Joining Year |
| 2 | 2015 |
| 1 | 2014 |

1. How clustered index works internally?
2. What is Strongly Typed view in mvc?
3. Diff between Exception Filter and try catch block? Both are used to handle exception?
4. Active directory in asp.net. LDAP/UDAP

### SQLCommand methods

ExecuteScalar() only returns the value from the first column of the first row of your query.

ExecuteReader() returns an object that can iterate over the entire result set.

ExecuteNonQuery() does not return data at all: only the number of rows affected by an insert, update, or delete.

### What is Data Readers ?

The **DataReader** object in C# ADO.NET allows you to retrieve data from database in read-only and forward-only mode. It means you can only read and display data but can’t update or delete data. If you want to make modification in retrieved data you need to use DataAdapter instead of DataReader.

### Why and when to use Data Readers ?

When you want to only display information or search result, you can use DataReader. There are various advantages of using DataReader like:

**1.** The retrieved data is stored in the network buffer in the client and then the client can read data using Read method. As data gets stored in the client network buffer it increases application performance significantly.

**2.** By default **DataReader** stores only one row at a time in memory. It reduces system overhead.

### METHODS AND PROPERTIES OF DATAREADER

#### Properties

|  |  |
| --- | --- |
| **PROPERTY** | **DESCRIPTION** |
| Depth | Indicates the depth of nesting for row |
| FieldCount | Returns number of columns in a row |
| IsClosed | Indicates whether a data reader is closed |
| Item | Gets the value of a column in native format |
| RecordsAffected | Number of row affected after a transaction |

#### Methods

|  |  |
| --- | --- |
| **METHOD** | **DESCRIPTION** |
| Close | Closes a DataRaeder object. |
| Read | Reads next record in the data reader. |
| NextResult | Advances the data reader to the next result during batch transactions. |
| Getxxx | There are dozens of Getxxx methods. These methods read a specific data type value from a  column. For example. GetChar will return a column value as a character and GetString as a  string. |

### HOW TO USE DATAREADERS IN ADO.NET FOR RETRIEVING DATA?

1. using **System**;
2. using **System**.**Data**.**SqlClient**;
4. namespace **DataReader\_Examples**
5. {
6. class **Program**
7. {
8. static void **Main**(string[] args)
9. {
10. string **ConString** = @"Data Source=.\SQLEXPRESS;Initial Catalog=ComputerShop;Integrated Security=True";
11. **SqlConnection** con = new **SqlConnection**(**ConString**);
12. string querystring = "Select \* from Items";
13. try
14. {
15. con.**Open**();
16. **SqlCommand** cmd = new **SqlCommand**(querystring, con);
17. **SqlDataReader** reader = cmd.**ExecuteReader**();
18. while (reader.**Read**())
19. {
20. **Console**.**WriteLine**(reader[0].**ToString**() + " " + reader[1].**ToString**() + " " + reader[2].**ToString**());
21. }
23. }
24. catch(**SqlException** ex)
25. {
26. **Console**.**WriteLine**(ex.**ToString**());
27. }
28. finally
29. {
30. con.**Close**();
31. **Console**.**ReadKey**();
32. }
33. }
34. }
35. }

**Output**

1 LED Screen $120 2 USB Keyboard $20 **\_**

### SQL Server Indexes

Heap Table:

SQL Server table can be organized within each partition in two ways: in Heap or B-Tree Clustered tables. In the **Heap** table, the data rows are not stored in any particular order within each data page.

Although the heap has no index structure that manages the pages and the data allocation, SQL Server Engine uses an **Index Allocation Map** (IAM) to keep an entry for each page to track the allocation of these available pages.

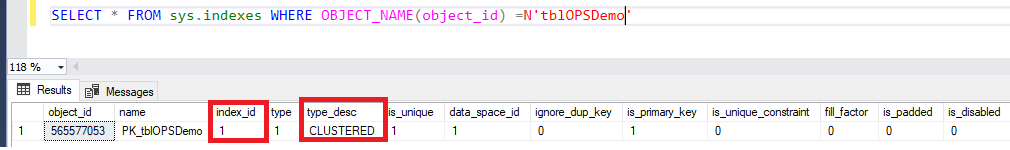
To perform a table scan on the heap table, SQL Server Engine will scan the IAM pages serially to locate the extents that are holding the requested data.

It is better not to keep the table, with no sorting mechanism, when you have large tables that you use to retrieve data from in specific sorting or grouping order as that will result in very bad performance. To avoid such performance issues, the table can be designed with internal ordering logic. This can be achieved by converting the table from heap table to a clustered table.

### Clustered Index

A clustered table is a table that has a predefined clustered index on one column of the table that defines the storing order of the rows within the data pages and the order of the pages within the table, based on the clustered index key. As the table rows can be stored only in single order, you can define only one clustered index on each table.

A clustered index is built using the B-tree structure, with one B-tree per each partition of the clustered table, in which the data pages in each level of the clustered index, from the root level until the leaf level, are linked in a doubly-linked list.



### Difference between Heap table and Clustered Index

Heap table structure, in which the data pages are not sorted in any ordering criteria and the pages itself are not sorted or linked between each other.

Clustered tables, in which the data is sorted within the data pages and the pages will be also linked in a double linked list, based on the index key.

<https://www.sqlshack.com/sql-server-index-structure-and-concepts/>

=============================================================================

### ACID

**ACID** (an acronym for Atomicity, Consistency Isolation, Durability) is a concept that Database Professionals generally look for when evaluating databases and application architectures. For a reliable database all these four attributes should be achieved.

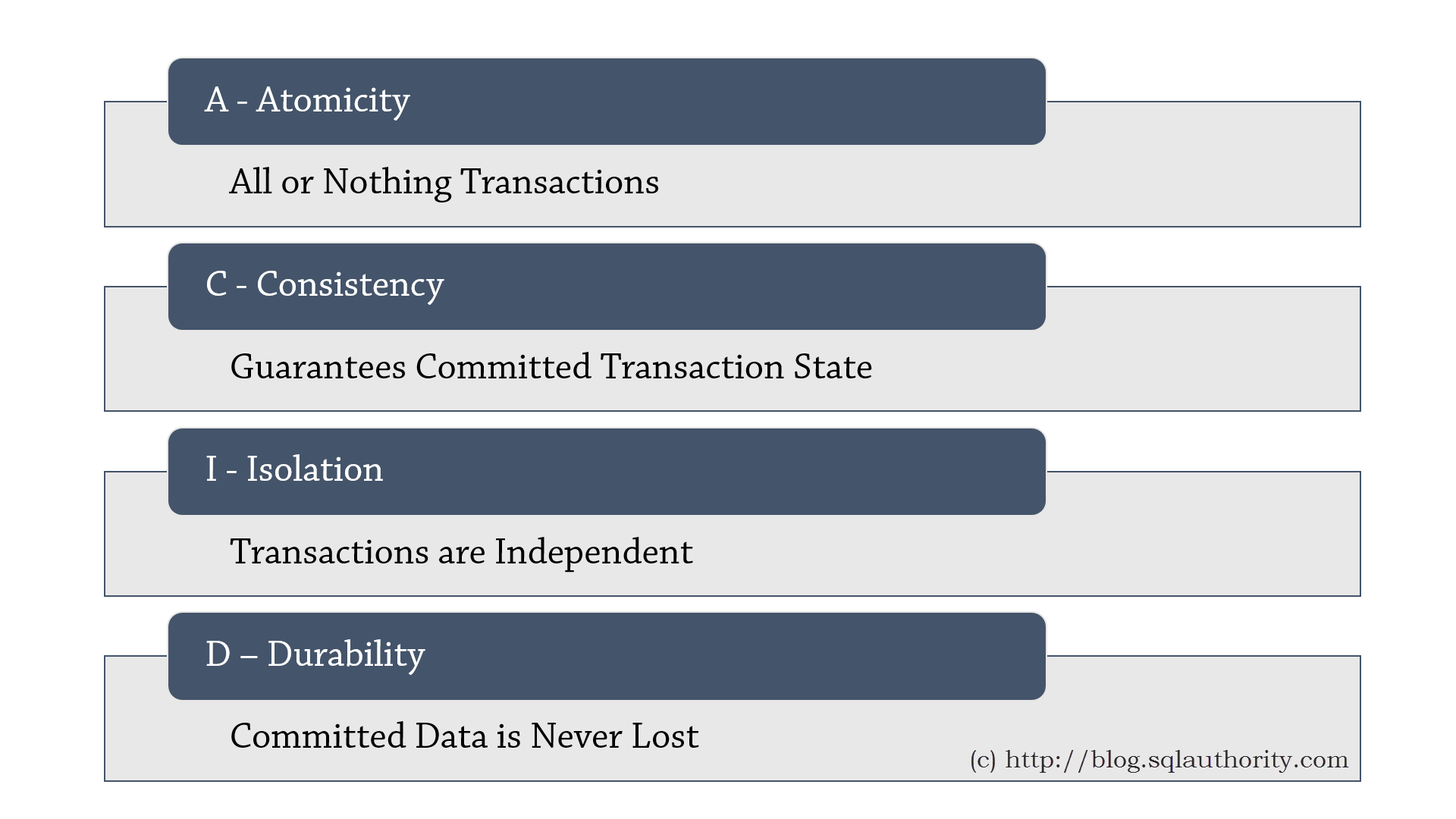
**Atomicity** is an all-or-none proposition.

**Consistency** guarantees that a transaction never leaves your database in a half-finished state.

**Isolation** keeps transactions separated from each other until they’re finished.

**Durability** guarantees that the database will keep track of pending changes in such a way that the server can recover from an abnormal termination.

Above four rules are very important for any developers dealing with databases.



### Transactions

A transaction is a single unit of work. If a transaction is successful, all of the data modifications made during the transaction are committed and become a permanent part of the database. If a transaction encounters errors and must be canceled or rolled back, then all of the data modifications are erased.

SQL Server operates in the following transaction modes:

**Autocommit transactions**  
Each individual statement is a transaction.

**Explicit transactions**  
Each transaction is explicitly started with the BEGIN TRANSACTION statement and explicitly ended with a COMMIT or ROLLBACK statement.

**Implicit transactions**  
A new transaction is implicitly started when the prior transaction completes, but each transaction is explicitly completed with a COMMIT or ROLLBACK statement.

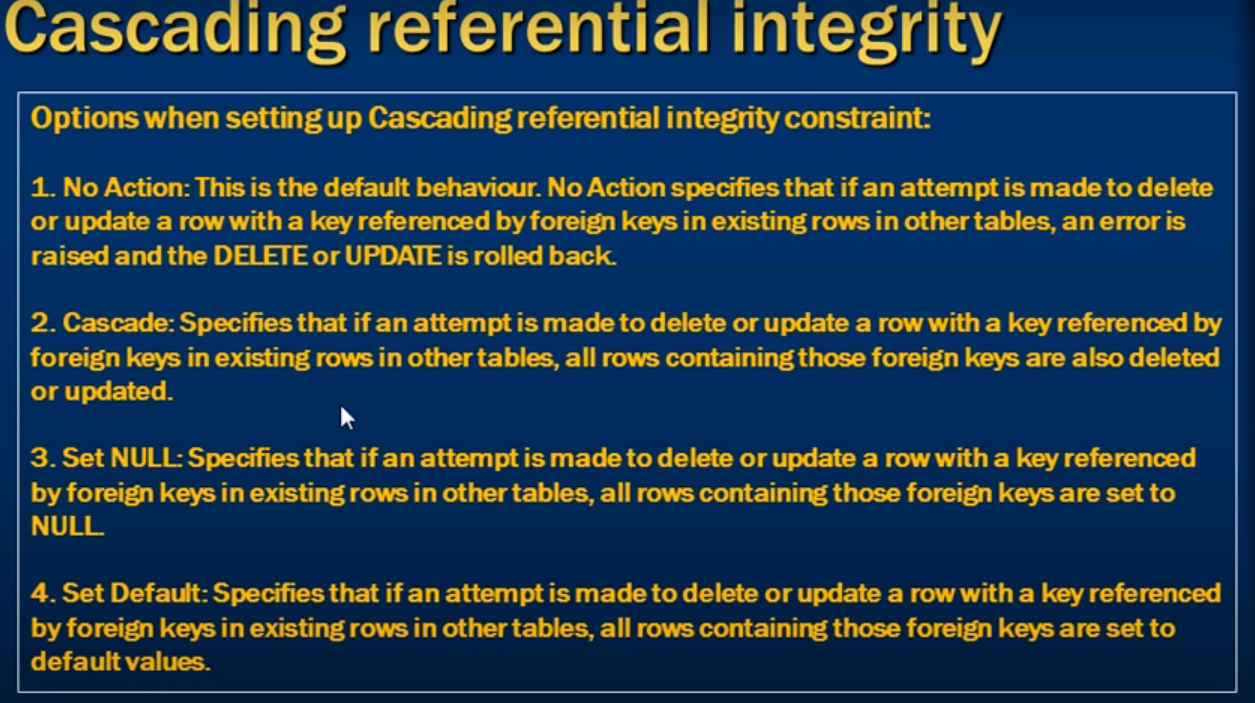
### Difference between DELETE, TRUNCATE and DROP

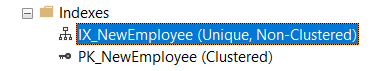
DELETE can be recovered from log file always if full recovery mode is set for database. TRUNCATE may or may not be recovered always from log files.

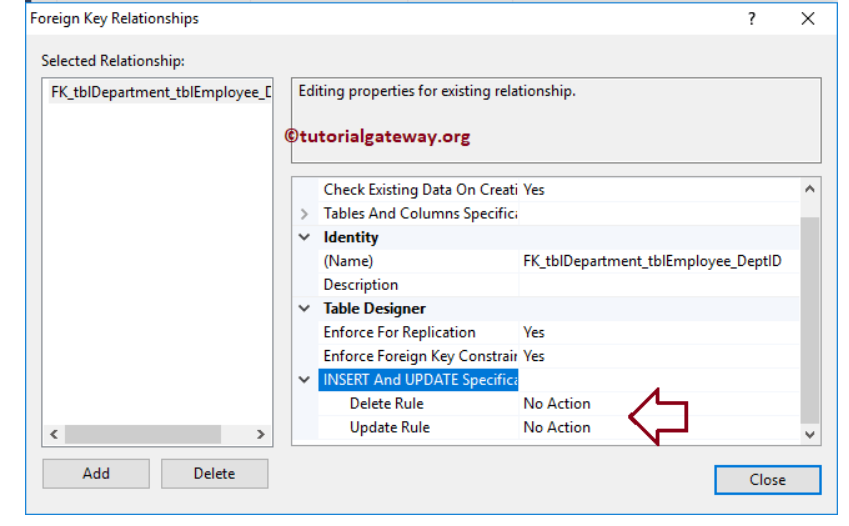
### What is Cascading referential integrity constraint?

Cascading referential integrity constraints are **foreign key constraints** which allow to perform certain actions when a user attempts to **delete or update a primary key** for which foreign keys exist.

Cascading Referential integrity applies to the **DELETE and UPDATE statements only** because they cause changes to existing rows.







## LINQ

### What is difference between IEnumerable and IQueryable?

**IEnumerable** executes the select query at the database and filters the data in-memory at the application layer.

**IQueryable** executes the select query and all of the filters at the database.

The database filtering reduced network traffic and application memory load resulting in a significant 10x performance gain.

|  | **IEnumerable** | **IQueryable** |
| --- | --- | --- |
| *Namespace* | System.Collections Namespace | System.Linq Namespace |
| *Derives from* | No base interface | Derives from IEnumerable |
| *[Deferred Execution](http://synvistech.com/blogs/deferred-execution-vs-lazy-loading-vs-eager-loading-vs-explicitly-loading/" \t "_blank" \o "Deferred Execution)* | Supported | Supported |
| *[Lazy Loading](http://synvistech.com/blogs/deferred-execution-vs-lazy-loading-vs-eager-loading-vs-explicitly-loading/" \t "_blank" \o "Lazy Loading)* | Not Supported | Supported |
| *How does it work* | While querying data from database, IEnumerable executes select query on server side, load data in-memory on client side and then filter data. Hence does more work and becomes slow. | While querying data from database, IQueryable executes select query on server side with all filters. Hence does less work and becomes fast. |
| *Suitable for* | LINQ to Object and LINQ to XML queries | LINQ to SQL queries |
| *Custom Query* | Doesn’t support | Supports using CreateQuery and Executemethods |
| *Extension method* *parameter* | Extension methods supported in IEnumerable takes functional objects. | Extension methods supported in IEnumerable takes expression objects, i.e., expression tree. |
| *When to use* | When querying data from in-memory collections like List, Array, etc. | When querying data from out-memory (like remote database, service) collections. |
| *Best Uses* | In-memory traversal | Paging |

**Except**: query operator returns the differences between the two collections

**Union**: query operator returns the unique elements from two collections?

**Intersect**: query operator returns the common elements from two collections?

**True statement:**

1. An Expression compiles into expression tree.
2. Expression tree is In-memory representation of lambda expression.
3. ‘To’ operators executes LINQ query immediately and gets the result.
4. Third party data providers must implement **IQuerable** interface, in order to support LINQ.

## Authentication in WebAPI

**[Secure a Web API with Individual Accounts and Local Login in ASP.NET Web API]**

**OR**

**[Token Based Authentication in ASP.NET Web API (Bearer authentication)]**

Web API project template gives you options for authentication:

* **Individual accounts.** The app uses a membership database.

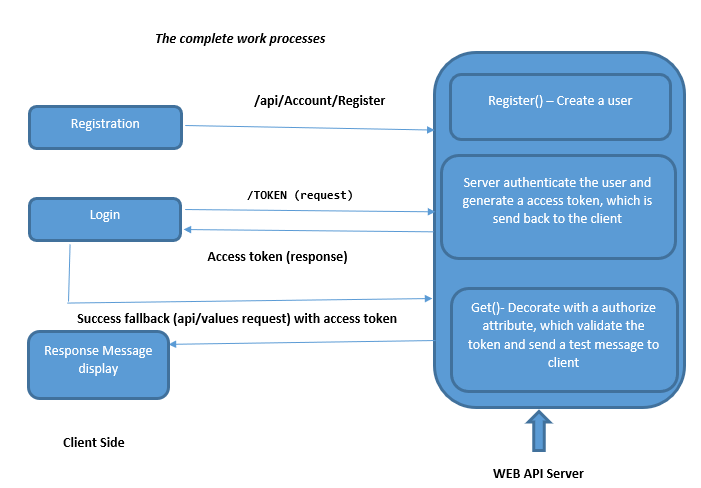
Individual accounts provide two ways for a user to log in:

* **Local login**. The user registers at the site, entering a username and password. The app stores the password hash in the membership database. When the user logs in, the ASP.NET Identity system verifies the password.

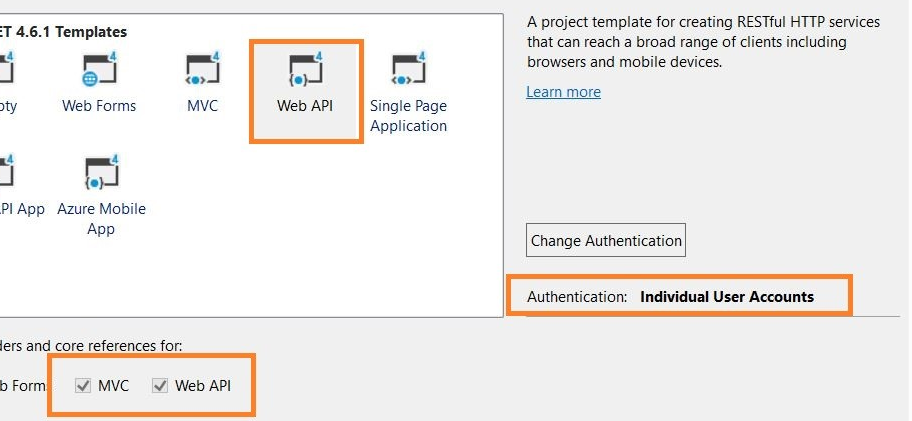
This article looks at the local login scenario. For both local and social login, **Web API uses OAuth2** to authenticate requests. However, the credential flows are different for local and social login.

### Token-based authentication

Token-based authentication is a process where the user sends his credential to the server, server will validate the user details and generate a token which is sent as response to the users, and user store the token in client side, so client do further HTTP call using this token which can be added to the header and server validates the token and send a response.



**STEP -1**

Open visual studio 2017 => create a new Web API project => Name the project, in my case, I named it as Token\_Auth\_Web\_API, set the Authentication to Individual User Account as shown in below figure. 

**STEP -2**

Go to Startup.cs file under App\_Start folder in the solution

// Configure the application for OAuth based flow

PublicClientId = "self";

OAuthOptions = new OAuthAuthorizationServerOptions

{

TokenEndpointPath = new PathString("/Token"),

Provider = new ApplicationOAuthProvider(PublicClientId),

AuthorizeEndpointPath = new PathString("/api/Account/ExternalLogin"),

AccessTokenExpireTimeSpan = TimeSpan.FromDays(14),

// In production mode set AllowInsecureHttp = false

AllowInsecureHttp = true

};

Install the **Owin** using the below command in package manager console

**Install-Package Owin -Version 1.0.0**

**Owin:**open web interface for .NET is a middleware which defines the interface between the web server and client application.

**TokenEndPointPath:**This is a kind of request path client applications which communicate with the server directly as part of the OAuth protocol. It must begin with slash “/”

1. **Provider:**The object provided by the application to process the event raised by the authorization server middleware.
2. **AuthorizeEndpointPath:**The request path where the client application will redirect the client/user to obtain user account to issue a token
3. **AccessTokenExpireTimeSpan :**Defines the validity of token
4. **AllowInsecureHttp:** It will allow a normal http request to authorize, if it is set to false, it will process only https request.

// GET api/values

[EnableCors(origins: "\*", headers: "\*", methods: "\*", exposedHeaders: "X-My-Header")]

[Authorize]

public IEnumerable<string> Get()

{

return new string[] {"You are successfully Authenticated to Access the Service"};

}

**[EnableCors(origins: "\*", headers: "\*", methods: "\*", exposedHeaders: "X-My-Header")] :** Enabled the CROS origin, so that it can be accessed from any domain

**[Authorize] :** It is used to authenticate the token send from the client side, once the authentication is successfully the Get() will be fired

### Client-side HTTP request with Authorization Header

$("#btnLogin").on('click', function () {

$.ajax(

{

url: "/TOKEN",

type: "POST",

data: $.param({ grant\_type: 'password', username: $("#loginEmail").val(), password: $("#loginPwd").val() }),

headers: { 'Content-Type': 'application/x-www-form-urlencoded' },

success: function (resp) {

sessionStorage.setItem('userName', resp.userName);

sessionStorage.setItem('accessToken', resp.access\_token);

var authHeaders = {};

authHeaders.Authorization = 'Bearer ' + resp.access\_token;

$.ajax({

url: "http://localhost:49501/api/values",

type: "GET",

headers: authHeaders,

success: function (response) {

$("#loginEmail").val("");

$("#loginPwd").val("");

$("#msg").text(response);

}

});

},

error: function () {

$("#msg").text("Authentication failed");

}

})

**authHeaders.Authorization = 'Bearer ' + resp.access\_token :** We are defining the authorization header with the access token when the /api/values HTTP call happens. In server side, the token is validated, once its success it will return a message “You are successfully Authenticated to Access the Service”

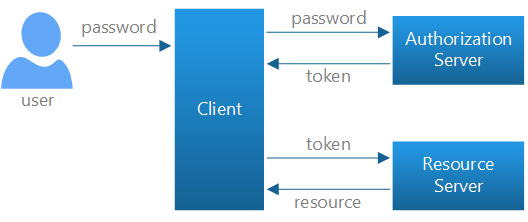
First, we need to define some **OAuth2 terminology**.

* *Resource*. Some piece of data that can be protected.
* *Resource server*. The server that hosts the resource.
* *Resource owner*. The entity that can grant permission to access a resource. (Typically the user.)
* *Client*: The app that wants access to the resource. In this article, the client is a web browser.
* *Access token*. A token that grants access to a resource.
* *Bearer token*. A particular type of access token, with the property that anyone can use the token. In other words, a client doesn't need a cryptographic key or other secret to use a bearer token. For that reason, bearer tokens should only be used over a HTTPS, and should have relatively short expiration times.
* *Authorization server*. A server that gives out access tokens.

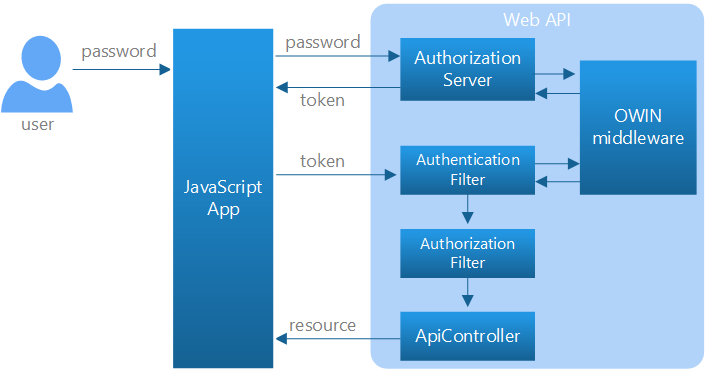
### Local Login Credential Flow

For local login, Web API uses the [resource owner password flow](http://oauthlib.readthedocs.org/en/latest/oauth2/grants/password.html) defined in OAuth2.

1. The user enters a name and password into the client.
2. The client sends these credentials to the authorization server.
3. The authorization server authenticates the credentials and returns an access token.
4. To access a protected resource, the client includes the access token in the Authorization header of the HTTP request.



When you select **Individual accounts** in the Web API project template, the project includes an authorization server that validates user credentials and issues tokens. The following diagram shows the same credential flow in terms of Web API components.



In this scenario, Web API controllers act as resource servers. An authentication filter validates access tokens, and the **[Authorize]** attribute is used to protect a resource. When a controller or action has the **[Authorize]** attribute, all requests to that controller or action must be authenticated. Otherwise, authorization is denied, and Web API returns a 401 (Unauthorized) error.

### [OWIN middleware](https://docs.microsoft.com/en-us/aspnet/aspnet/overview/owin-and-katana/an-overview-of-project-katana)

The authorization server and the authentication filter both call into an [OWIN middleware](https://docs.microsoft.com/en-us/aspnet/aspnet/overview/owin-and-katana/an-overview-of-project-katana) component that handles the details of OAuth2.

Open Web Interface for.NET (**OWIN**) is an open-source specification that describes an abstraction layer between web servers and application components.

**OAuth**

**OAuth** is an open standard for Authorization, which is a token-based authorization.

Using OAuth, users of a web site or an app can be authenticated by a third party login-password (like Microsoft Live, LinkedIn, Facebook, google) etc. and then a token is returned to your site/app as a representation that authentication has been done.

Then, you can use this token to give access to your authenticated users to some resources.

While on the other hand OWin is Open Web Interface for .Net. Using OWIN, the tight coupling between a dot net web application  and the server on which it is hosted can be eliminated. It helps eliminate tight coupling between ASP.Net and IIS.

### [RESTful](https://docs.microsoft.com/en-us/aspnet/aspnet/overview/owin-and-katana/an-overview-of-project-katana) Service

**Stateful:** If the webserver stores data in a backend manner and uses it to identify the user as an always-connected client, the service is **Stateful**.

**Stateless**: the server does store data, but in a database to verify user/client whenever it needs to connect.

**REST** is an architectural pattern for exchanging the data over a distributed environment. REST architectural pattern treats each **service as a resource** and a client can access these resources by using HTTP protocol methods like GET, POST, PUT, and DELETE.

The REST architectural pattern specifies a set of constraints that a system should adhere to.

Client-Server constraint

Stateless constraint

Cacheable constraint

Uniform Interface constraint:

The HTTP verb (GET, PUT, POST, and DELETE) that is sent with each request tells the API what to do with the resource. Each resource is identified by a specific URI (Uniform Resource Identifier).

### WCF Rest vs Web API?

**WCF Rest**

WebHttpBinding to be enabled for WCF Rest.

For each method, there have to be attributes like – “WebGet” and “WebInvoke”

For GET and POST verbs respectively.

**Web API**

Unlike WCF Rest we can use full features of HTTP in Web API.

Web API can be hosted in IIS or in an application.

### WCF Rest and Web API?

### WCF and Web API?

### New features of Web API 2.0?

The new features introduced in ASP.NET Web API framework v2.0 are as follows:

1. Attribute Routing
2. External Authentication
3. CORS (Cross-Origin Resource Sharing)
4. OWIN (Open Web Interface for .NET) Self Hosting
5. IHttpActionResult
6. Web API OData

## Action Result Type in WEB API

A Web API controller action can return any of the following:

1. void
2. **HttpResponseMessage**
3. **IHttpActionResult**
4. Some other type

Depending on which of these is returned, Web API uses a different mechanism to create the HTTP response.

|  | |
| --- | --- |
| **Return type** | **How Web API creates the response** |
| Void | Return empty 204 (No Content) |
| HttpResponseMessage | Convert directly to an HTTP response message. |
| IHttpActionResult | Call **ExecuteAsync** to create an **HttpResponseMessage**, then convert to an HTTP response message. |
| Other type | Write the serialized return value into the response body; return 200 (OK). |

### Void Return Type

If the return type is void, Web API simply returns an empty HTTP response with status code 204 (No Content).

Example controller:

C#Copy

public class ValuesController : ApiController

{

public void Post()

{

}

}

HTTP response:

ConsoleCopy

HTTP/1.1 204 No Content

Server: Microsoft-IIS/8.0

Date: Mon, 27 Jan 2014 02:13:26 GMT

### HttpResponseMessage

If the action returns an [HttpResponseMessage](https://msdn.microsoft.com/library/system.net.http.httpresponsemessage.aspx), Web API converts the return value directly into an HTTP response message, using the properties of the **HttpResponseMessage** object to populate the response.

This option gives you a lot of control over the response message. For example, the following controller action sets the Cache-Control header.

C#Copy

public class ValuesController : ApiController

{

public HttpResponseMessage Get()

{

HttpResponseMessage response = Request.CreateResponse(HttpStatusCode.OK, "value");

response.Content = new StringContent("hello", Encoding.Unicode);

response.Headers.CacheControl = new CacheControlHeaderValue()

{

MaxAge = TimeSpan.FromMinutes(20)

};

return response;

}

}

Response:

ConsoleCopy

HTTP/1.1 200 OK

Cache-Control: max-age=1200

Content-Length: 10

Content-Type: text/plain; charset=utf-16

Server: Microsoft-IIS/8.0

Date: Mon, 27 Jan 2014 08:53:35 GMT

hello

If you pass a domain model to the **CreateResponse** method, Web API uses a [media formatter](https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/media-formatters) to write the serialized model into the response body.

C#Copy

public HttpResponseMessage Get()

{

// Get a list of products from a database.

IEnumerable<Product> products = GetProductsFromDB();

// Write the list to the response body.

HttpResponseMessage response = Request.CreateResponse(HttpStatusCode.OK, products);

return response;

}

Web API uses the Accept header in the request to choose the formatter. For more information, see [Content Negotiation](https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/content-negotiation).

### IHttpActionResult

**IHttpActionResult** contains a single method, **ExecuteAsync**, which asynchronously creates an **HttpResponseMessage** instance.

public interface IHttpActionResult

{

Task<HttpResponseMessage> ExecuteAsync(CancellationToken cancellationToken);

}

If a controller action returns an **IHttpActionResult**, Web API calls the **ExecuteAsync** method to create an **HttpResponseMessage**. Then it converts the **HttpResponseMessage** into an HTTP response message.

The **IHttpActionResult** interface was introduced in Web API 2. Essentially, it defines an **HttpResponseMessage** factory. Here are some advantages of using the **IHttpActionResult** interface:

* Simplifies [unit testing](https://docs.microsoft.com/en-us/aspnet/web-api/overview/testing-and-debugging/unit-testing-controllers-in-web-api) your controllers.
* Moves common logic for creating HTTP responses into separate classes.
* Makes the intent of the controller action clearer, by hiding the low-level details of constructing the response.

Here is a simple implementation of **IHttpActionResult** that creates a plain text response:

C#Copy

public class TextResult : IHttpActionResult

{

string \_value;

HttpRequestMessage \_request;

public TextResult(string value, HttpRequestMessage request)

{

\_value = value;

\_request = request;

}

public Task<HttpResponseMessage> ExecuteAsync(CancellationToken cancellationToken)

{

var response = new HttpResponseMessage()

{

Content = new StringContent(\_value),

RequestMessage = \_request

};

return Task.FromResult(response);

}

}

Example controller action:

C#Copy

public class ValuesController : ApiController

{

public IHttpActionResult Get()

{

return new TextResult("hello", Request);

}

}

Response:

HTTP/1.1 200 OK

Content-Length: 5

Content-Type: text/plain; charset=utf-8

Server: Microsoft-IIS/8.0

Date: Mon, 27 Jan 2014 08:53:35 GMT

hello

More often, you use the **IHttpActionResult** implementations defined in the **[System.Web.Http.Results](https://msdn.microsoft.com/library/system.web.http.results.aspx)** namespace. The **ApiController** class defines helper methods that return these built-in action results.

In the following example, if the request does not match an existing product ID, the controller calls [ApiController.NotFound](https://msdn.microsoft.com/library/system.web.http.apicontroller.notfound.aspx) to create a 404 (Not Found) response. Otherwise, the controller calls [ApiController.OK](https://msdn.microsoft.com/library/dn314591.aspx), which creates a 200 (OK) response that contains the product.

C#Copy

public IHttpActionResult Get (int id)

{

Product product = \_repository.Get (id);

if (product == null)

{

return NotFound(); // Returns a NotFoundResult

}

return Ok(product); // Returns an OkNegotiatedContentResult

}

### Other Return Type

For all other return types, Web API uses a [media formatter](https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/media-formatters) to serialize the return value. Web API writes the serialized value into the response body. The response status code is 200 (OK).

C#Copy

public class ProductsController : ApiController

{

public IEnumerable<Product> Get()

{

return GetAllProductsFromDB();

}

}

A disadvantage of this approach is that you cannot directly return an error code, such as 404. However, you can throw an **HttpResponseException** for error codes. For more information, see [Exception Handling in ASP.NET Web API](https://docs.microsoft.com/en-us/aspnet/web-api/overview/error-handling/exception-handling).

Web API uses the Accept header in the request to choose the formatter. For more information, see [Content Negotiation](https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/content-negotiation).

Example **Request**

GET http://localhost/api/products HTTP/1.1

User-Agent: Fiddler

Host: localhost:24127

Accept: application/json

Example **Response**

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Server: Microsoft-IIS/8.0

Date: Mon, 27 Jan 2014 08:53:35 GMT

Content-Length: 56

[{"Id":1,"Name":"Yo-yo","Category":"Toys","Price":6.95}]

### Content Negotiation in WEB API

In which format client wants the **response** – whether it is XML or JSON. This is called **Content Negotiation**.

* **Accept:** Which media types are acceptable for the response, such as "application/json,"

"application/xml,"

or a custom media type such as "application/vnd.example+xml"

**Difference between Accept Header and Content Type**

**Accept**: Used inside **Request** Body

**Content-Type**: Used inside **Response** Body

**Request Body**

GET http://localhost/api/products HTTP/1.1

User-Agent: Fiddler

Host: localhost:24127

Accept: application/json

**Response Body**

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Server: Microsoft-IIS/8.0

Date: Mon, 27 Jan 2014 08:53:35 GMT

Content-Length: 56

[{"Id":1,"Name":"Yo-yo","Category":"Toys","Price":6.95}]

<https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/content-negotiation>

### Media Type Formatter in WEB API

### IHttpActionResult Vs HttpResponseMessage

<https://www.youtube.com/watch?v=cNR7gDPJtQI>

<https://www.c-sharpcorner.com/blogs/a-new-way-to-send-response-using-ihttpactionresult1>

* How to return a HTTP Response from the controller with a single line of code using the interface. Have a look at the following example.

public class personController : ApiController

{

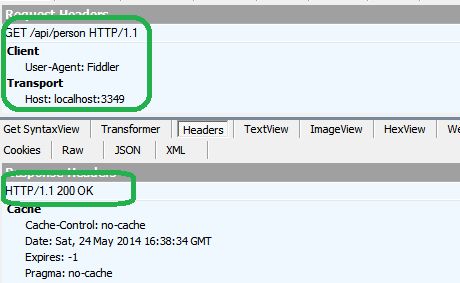
    public IHttpActionResult Get()

    {

        return Ok();

    }

}



* How to embed some value with the HTTP response message? Fine, the next example is for you.

public class personController : ApiController

{

    public IHttpActionResult Get()

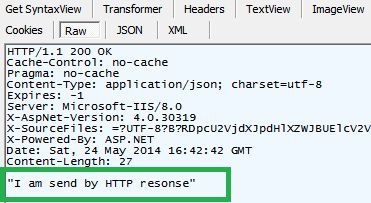
    {

        return Ok<string>("I am send by HTTP resonse");

    }

}

We are just returning string a in the message body and the output in Fiddler is something like this.



* Not only string, we can send any complex type of custom data type as a body of the HTTP response message. In the next example we will try to send a list of strings in the body of the response message with an Ok status. Here is our modified code.

public IHttpActionResult Get()

{

    List<string> names = new List<string> {

       "Sourav",

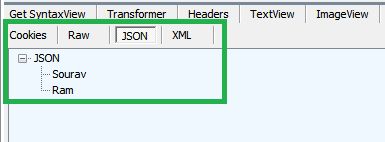
       "Ram"

    };

    return Ok<List<string>> (names);

}   

And the output is in JSON format.



* Not only an Ok message, but we can also return any type of valid HTTP response, let's see a few of them.  
    
  **Not Found**

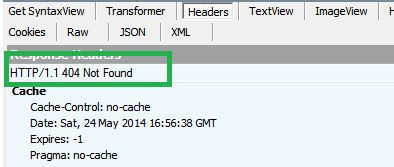
public IHttpActionResult Get()

{

   return NotFound();

}

Like Ok() , we can return NotFound() , it will return a 404 status code as in the following screen.

  
  
**Bad Request**

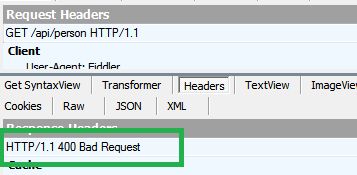
public IHttpActionResult Get()

{

   return BadRequest();

} 

We know the status code for BadRequest is 400 and once we call the method, we will get the status.

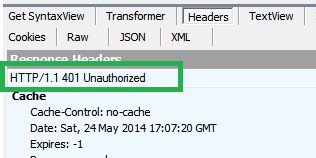
  
  
**Unauthorized**  
  
In the same way, we can return an unauthorized status code, here is sample code.

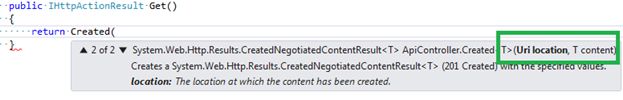
public IHttpActionResult Get()

{

   return Unauthorized();

} 

****

* **Created**  
    
  The status code for the Created status is 201 and generally the status is returned when the Post() operation is performed successfully. Created takes two parameters, one is the "uri" and the other is content.  
    
  

## Points to remember in C#

* By default, access modifier of class is internal in C#.
* We can make root class only public and internal in C#. Means we cannot declare class as private, protected, protected internal, private protected.
* By default, nested class is private in C#. We cannot access nested class outside when it is set private and protected.
* You know C# is case sensitive language So you can not change the name of “Main” method name. For Example: you can not use like this: “main”,”MaiN” etc
* Private constructor is required because it is not allowed the class to be instantiated from outside the class. It only instantiates from within the class members.

(Its members: fields, properties and methods). Whenever its member is called then private constructor will automatically called.

* Class with private constructor can not be inherited outside but can be inherited with nested class.

<https://www.c-sharpcorner.com/uploadfile/Ashush/asynchronous-nature-of-delegates/>

## Delegates

Delegates are the **Type-Safe Function Pointer.**It means they hold the reference of a method and then calls that method for execution.

We can **call/Invoke a method** in C# by using delegates. Calling a C# method using delegate will be **faster in execution** as compared to the first process i.e. either by using an object or by using the class name.

If you want to invoke or call a method using delegates then you need to follow three simple steps.  The steps are as follows.

1. **Defining a delegate**
2. **Instantiating a delegate**
3. **Invoking a delegate**
4. **Defining a delegate**

<Access Modifier> delegate <return type> <delegate name> (arguments list);

**Example:**If you have a method like below.

**public** **void** Add**(int** x, **int** y**)**

**{**

Console.WriteLine**(**@"The Sum of {0} and {1}, is {2} ", x, y, **(**x + y**))**;

**}**

**Then you have to define a delegate like public delegate void AddDelegate(int a, int b);**

1. **Instantiating a delegate**

<Access Modifier> delegate <return type> <delegate name> (arguments list);

**Syntax: Delegate Name Object Name = new Delegate Name (target function name);**

**Example:**

**AddDelegate ad = new AddDelegate(obj.Add);**

1. **Invoking the delegate**

Now call the delegate by supplying the required values to the parameters so that the methods get executed internally which is bound with the delegates.

**For example:**

**ad(100,50);**   
**ad.Invoke(200, 300);**

**(Or we can use Invoke method by using Invoke keyword)**

**Points to remember about delegate:**

* The access modifier, return type and the number, order and type of parameters of delegate **should be same as** the method it refers to. This is the reason why delegates are called as **type-safe function pointers**.
* Methods are passed as a **parameter** in the delegate constructor.
* Delegates can **invoke more than one method** using the Multicast feature.
* Delegate instances **attach or detach a method at run time** making it more dynamic and flexible to use. Use **+=** operator to add methods in delegate to chain together and **-=** operator to remove the method that you want.
* **Complier** is unaware of the information of which the method will be called by delegate. The information will be given at **runtime** by the delegates instance constructor that is very flexible to use.
* Delegates are of **reference types.**
* A Delegate is a **type-safe object**.
* A Delegate is **invoking to methods** in **asynchronously** manner.
* **Encapsulate** to methods call from caller(Main Method).

### Where to use delegate / Main objective to use delegate:

The following are the lists of areas where a delegate can be used to provide **better flexibility and improve the performance** of the application.

* **Method Invocation** (using delegate instance): Calling a method using delegate will be faster in execution as compared to either by using an object or by using the class name. Complier is unaware of the information of which the method will be called by delegate. The information will be given at runtime by the delegates instance constructor that is very flexible to use.
* **Event Handling using delegate:**

**Example:** Window Form Application

**Form1.Designer.cs**

**this**.button1.Click += **new** System.EventHandler(**this**.button1\_Click);

**Form1.cs is,**

1. privatevoid button1\_Click(**object** sender, EventArgs e)
2. {
4. }

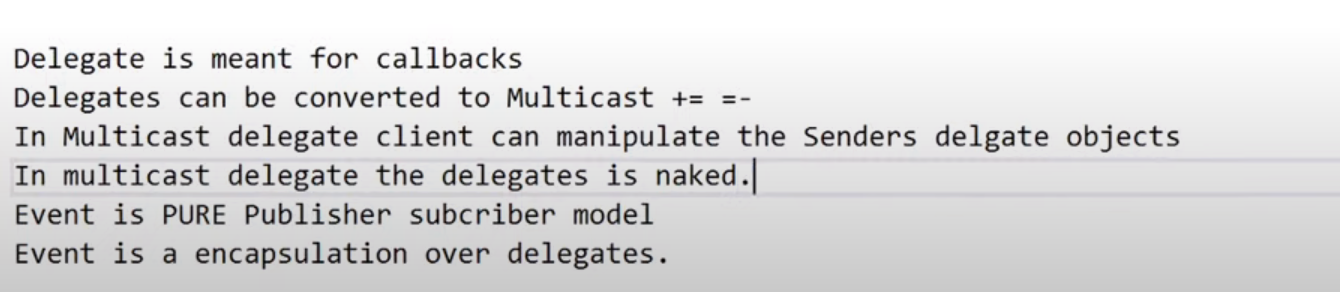
So you can see System.EventHandler is a delegate which is associating button1\_Click method with button1 click event.

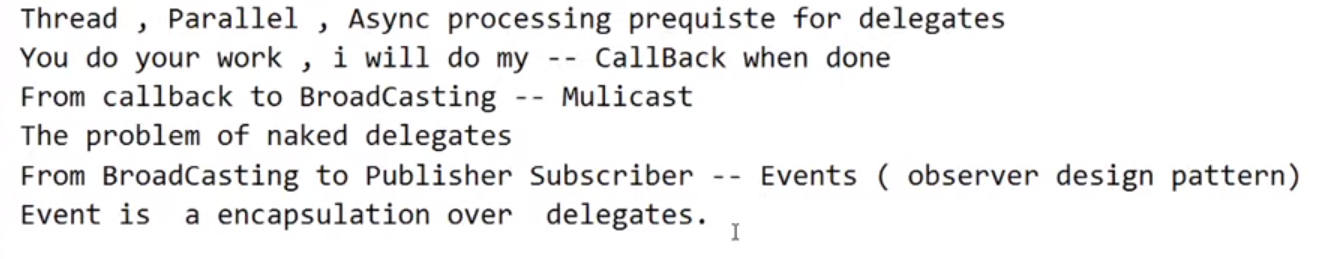
* **Call-back and asynchronous implementation:** Asynchronous Programming Using Delegates

1. When you use your delegate to call a method you actually call the Invoke() method. Invoke() method executes your method **synchronously**.
2. When you want to make an **asynchronous** call to your underlying method first you call **BeginInvoke().** When you call BeginInvoke() your method is queued to start on another thread.
3. BeginInvoke() doesn't return the return value of the underlying method. Instead it returns an **IAsyncResult** object that you can use to determine when the asynchronous operation is complete.
4. To take your results, you submit the **IAsyncResult** object to the **EndInvoke()** method of the delegate. **EndInvoke()** waits for the operation to complete if it hasn't already finished and then provide the return value.

* **Multiple method calls using Multicast delegate**

**Asynchronous** means two or more operations are running in different thread so that they can run concurrently and do not block each other.





### In-build Delegates

Microsoft provides three ready-made generics delegate. This is call shorter and sweeter delegate. This to minimize the code complexity of the apps.

1. Func< input, output >

2. Action< inputParameter >

3. Predicate< inputParameter >

**Func <input, output > Delegate:-**

The Func <input, output> generic delegate is use, when you need to some input parameter and then return some output.

**Action< inputParameter > Delegate:-**

The Action< inputParameter> generics delegate is use, when you need to input parameter and no need to return output that means return void.

**Predicate < inputParameter > Delegate:-**

The Predicate generics delegate is use, when you need to input parameter and return output.

The return output is Boolean type just like true or false. This delegate is uses when you need to check the conditional statements are true or false.

**The example for Generic Delegates:-**namespace GenericDelegate

{

    public class Delegates

    {

        static void Main(string[] args)

        {

            //Int is an input parameter and double is output parameter.

            Func<int, double > calPiR2Obj = r => 3.12 \* r \* r;

            Console.WriteLine(calPiR2Obj(4));

            //String is input but no output return.

            Action<string > ActionObject = x => Console.WriteLine(x);

            ActionObject("This is Action delegate!");

            //String is input but output, the return is true or false

            Predicate< string > checkConditionIsTF = x => x.Length > 10;

            Console.WriteLine(checkConditionIsTF("Anil Kumar Singh"));

            Console.ReadLine();

        }

    }

}

### Custom Delegates & Event Handling with Window Form Application

**In Form1.Designer.cs the code snippet is,**

1. //
2. // button1
3. //
4. **this**.button1.Location = **new** System.Drawing.Point(37, 26);
5. **this**.button1.Name = "button1";
6. **this**.button1.Size = **new** System.Drawing.Size(75, 23);
7. **this**.button1.TabIndex = 0;
8. **this**.button1.Text = "button1";
9. **this**.button1.UseVisualStyleBackColor = **true**;

**this**.button1.Click += **new** System.EventHandler(**this**.button1\_Click);

**And code snippet for Form1.cs is,**

1. privatevoid button1\_Click(**object** sender, EventArgs e)
2. {
4. }

So you can see System.EventHandler is a delegate which is associating button1\_Click method with button1 click event.  
  
If you press F12 (go to definition) on EventHandler then you will find the following code snippet.

1. **namespace** System
2. {
3. //
4. // Summary:
5. // Represents the method that will handle an event that has no event data.
6. //
7. // Parameters:
8. // sender:
9. // The source of the event.
10. //
11. // e:
12. // An object that contains no event data.
13. [ComVisible(**true**)]
14. Public delegate void EventHandler(**object** sender, EventArgs e);
15. }

### Delegates & Event Handling with ASP.NET Web Form

Now create an ASP.NET web Form application.  
  
**On .aspx page the following code snippet is written**

1. <formid="form1" runat="server">
2. <div>
4. <asp:ButtonID="Button1" runat="server" OnClick="Button1\_Click" Text="Button" />
6. </div>
7. </form>

**And on .cs file the following code snippet is written**

1. Protected void Button1\_Click(**object** sender, EventArgs e)
2. {
3. //do something
4. }

Here Button1\_Click written with OnClick="Button1\_Click" is delegate but you may be thinking that where the delegate is calling the method. Actually it is happening behind the scene. This is a syntactic sugar of ASP.NET web form.

### Asynchronous programming using Delegate

**Multithreaded application** can be done in several ways includes:

**1. Asynchronous delegates calls.**

**2. The BackgroundWorker component.**

**3. The Thread class.**

**Asynchronous delegates calls:**   
as you may know, we use the delegate to point to method in the application, which can be invoked at later time.

**Example 1:**

namespace DelegateSynchronousCalls

{

    public delegate int MyDelegate(int x);

    public class MyClass

    {

        //A method to be invoke by the delegate

        public int MyMethod(int x)

        {

            return x \* x;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            MyClass myClass1 = new MyClass();

            MyDelegate del = new MyDelegate(myClass1.MyMethod);

            //invoke the method synchronously

            int result = del(5);

            //this text will not show till the first operation finish

            Console.WriteLine("Proccessing operation...");

            Console.WriteLine("Result is: {0}", result);

            Console.ReadLine();

        }

    }  
}  
  
As you can see we used our delegate to invoke the method in the main thread of our application, but what if we have a long runing proccess that will be executed when we invoke the method, for example:  
  
public int MyMethod(int x)

{

    //simulate a long running proccess

    Thread.Sleep(10000);

    return x\*x;

}  
  
We have now an operation that will take about 10 second to complete so our application will be not responsive and we can't execute any other operations till the first one finish.  
  
So, what we can do to overcome this problem?  
  
The answer is to use the delegate to invoke the method asynchronosly, but how we can do this?. The next example will show you how to use our delegate to make asynchronous call of the method.  
  
**Example 2:**  
namespace DelegatesAsynchronousCalls

{

    class Program

    {

        public delegate int MyDelegate(int x);

        public class MyClass

        {

            //A method to be invoke by the delegate

            public int MyMethod(int x)

            {

                //simulate a long running proccess

                Thread.Sleep(10000);

                return x \* x;

            }

        }

        static void Main(string[] args)

        {

            MyClass myClass1 = new MyClass();

            MyDelegate del = new MyDelegate(myClass1.MyMethod);

            //Invoke our method in another thread

            IAsyncResult async = del.BeginInvoke(5, null, null);

            //do something while MyMethod is executing.

            Console.WriteLine("Proccessing operation...");

            //recieve the results.

            int result = del.EndInvoke(async);

            Console.WriteLine("Result is: {0}", result);

            Console.ReadLine();

        }

    }

}  
  
As you can see, we defined the delegate as usual but we don't start invoking our method directly, we used our delegate to call BeginInvoke() method that returns an IAsyncResult object, then we submit this object to the EndInvoke() method then EndInvoke() provide the return value.(complex right !!?)  
  
You may wonder now from where all this methods come from?  
  
The answer is simple, when you define a delegate, a custom delegate class is generated and added to your assembly. this class include many method like Invoke(), BeginInvoke() and EndInvoke().  
  
When you use your delegate to call a method you actualy call the Invoke() method. Invoke() method executes your method synchronously (on the main thread) as in example 1.  
  
When you want to make an asynchronous call to your underlying method first you call BeginInvoke(). When you call BeginInvoke() your method is queued to start on another thread.    
  
BeginInvoke() doesn't return the return value of the underlying method. Instead it returns an IAsyncResult object that you can use to determine when the asynchronous operation is complete.  
  
To take your results, you submit the IAsyncResult object to the EndInvoke() method of the delegate. EndInvoke() waits for the operation to complete if it hasn't already finished and then provide the return value.  
  
You may ask, what is the other two parameters in the BeginInvoke() method?  
  
When calling BeginInvoke() you supply all the parameters of the original method, plus two parameters, the first can be used in callback and the othe is a state object. if you don't need these options just pass a null reference.   
  
In the next section you will see how to use them.

**Using AsyncCallback delegate:**  
  
To allow the calling thread to know if the asynchronous operation has completed its work, you have two ways to do this, first you can make use of the IsCompleted property of the IAsyncResult interface.  
  
By using this property, the calling thread is able to determine is the asynchronous operation has completed before calling EndInvoke(). If the method has not completed, IsCompleted returns false. If IsCompleted return true, the calling thread can obtain the results  
  
**Example 3:**

namespace DelegatesAsyncCallsIsCompleted

{

    public delegate int MyDelegate(int x);

    public class MyClass

    {

        //A method to be invoke by the delegate

        public int MyMethod(int x)

        {

            //simulate a long running proccess

            Thread.Sleep(3000);

            return x \* x;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            MyClass myClass1 = new MyClass();

            MyDelegate del = new MyDelegate(myClass1.MyMethod);

            //Invoke our methe in another thread

            IAsyncResult async = del.BeginInvoke(5, null, null);

            //loop until the method is complete

            while (!async.IsCompleted)

            {

                Console.WriteLine("Not Completed");

            }

            int result = del.EndInvoke(async);

            Console.WriteLine("Result is: {0}", result);

            Console.ReadLine();

        }

    }

}  
  
It would be more efficient when we have a delegate that inform the calling thread when the operation is finished.  
  
To do this, you need to supply an instance of the AsyncCallback delegate as a parameter to BeginInvok(). This delegate will call a specified method automaticaly when the asynchronous call has completed.   
  
The method that AsyncCallback will invoke must taking IAsyncResult as a sole parameter and return nothing.   
  
public static void MyCallBack(IAsyncResult async)

{

}  
  
The IAsyncResult is the same object that you recieve when you call BeginInvoke() method.  
  
**Example 4:**  
namespace AsyncCallbackDelegate

{

    public delegate int MyDelegate(int x);

    public class MyClass

    {

        //A method to be invoke by the delegate

        public int MyMethod(int x)

        {

            //simulate a long running proccess

            Thread.Sleep(10000);

            return x \* x;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            MyClass myClass1 = new MyClass();

            MyDelegate del = new MyDelegate(myClass1.MyMethod);

            //Invoke our methe in another thread

            IAsyncResult async = del.BeginInvoke(5, new AsyncCallback(MyCallBack), null);

            Console.WriteLine("Proccessing the Operation....");

            Console.ReadLine();

        }

        static void MyCallBack(IAsyncResult async)

        {

            Console.WriteLine("Operation Complete);

        }

    }

}  
  
Remember, the MyCallback() method will be invoked by the AsyncCallback delegate when the MyMethod() has completed.  
As you can see, we didn't invoke the EndInvoke() method that because the MyCallback() method doesn't have access to the MyDelegate object, so we can make use of the incoming IAsyncResult parameter and cast it into type AsyncResult and use the static AsyncDelegate property which returns a reference to the original asynchronous delegate that was created else where.  
  
We can rewrite the MyCallback() method as follow:  
  
static void MyCallBack(IAsyncResult async)

{

    AsyncResult ar = (AsyncResult)async;

    MyDelegate del = (MyDelegate)ar.AsyncDelegate;

    int x = del.EndInvoke(async);

    Consol.WriteLine("Operation Complete, Result is: {0}", x);

}

The final parameter of the BeginInvoke() method can be used to pass additional state information to the callback method from the primary thread. Because this parameter is a System.Object, you can pass in any type of information .

==============================================================================

### Anonymous Method

An anonymous method is a method without having a name.

Anonymous methods can be defined using the keyword delegate and can be assigned to a variable of the delegate type.

* **Bind a code block to a delegate without having name is called Anonymous method**

**Example**:

namespace DelegateDemo

{

public class AnonymousMethods

{

public delegate string GreetingsDelegate(string name);

static void Main(string[] args)

{

GreetingsDelegate gd = delegate(string name)

{

return "Hello @" + name + " Welcome to Dotnet Tutorials";

};

string GreetingsMessage = gd.Invoke("Pranaya");

Console.WriteLine(GreetingsMessage);

Console.ReadKey();

}

}

}

##### **Points to Remember**

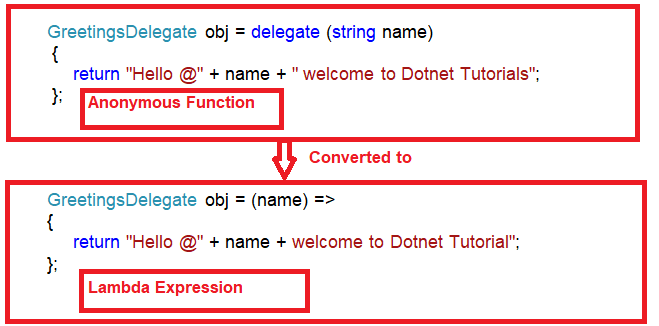
1. The anonymous methods can be **defined using the delegate** keyword
2. An anonymous method must be **assigned to a delegate type**.
3. This method **can access outer variables or functions**.
4. An anonymous method can be **passed as a parameter**.
5. This method can be **used as event handlers**.

### Lambda Expression

Lambda Expression **simplify the anonymous function.**

To create a lambda expression in C#, we need to specify the input parameters (if any) on the left side of the lambda operator **=>**, and we need to put the expression or statement block on the other side.

**Input parameters => Statement block**



**Let us rewrite the same example by using Lambda Expression in C#.**

**namespace** *LambdaExpressionDemo*

**{**

**public** **class** LambdaExpression

**{**

**public** **delegate** string GreetingsDelegate**(**string name**)**;

**static** **void** Main**(**string**[]** args**)**

**{**

GreetingsDelegate obj = **(**name**)** =**>**

**{**

**return** "Hello @" + name + " welcome to Dotnet Tutorials";

**}**;

string GreetingsMessage = obj.Invoke**(**"Pranaya"**)**;

Console.WriteLine**(**GreetingsMessage**)**;

Console.ReadKey**()**;

**}**

**public** **static** string Greetings**(**string name**)**

**{**

**return** "Hello @" + name + " welcome to Dotnet Tutorials";

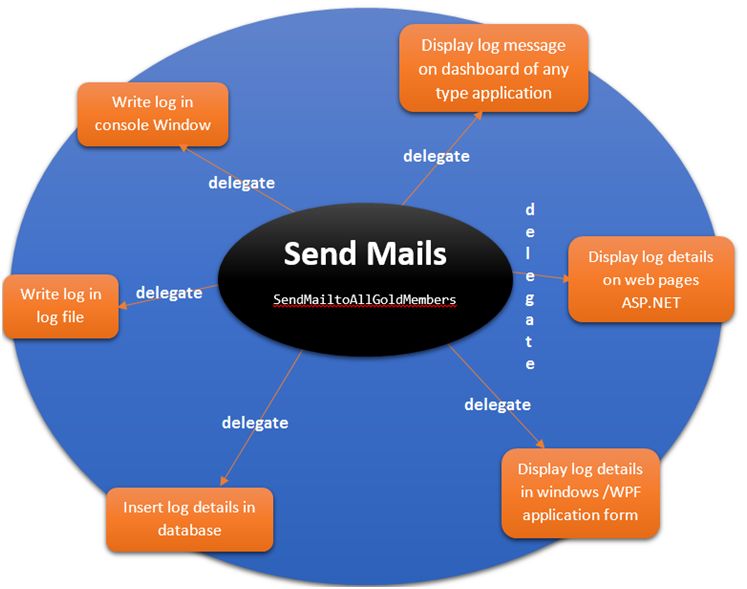
**}**

**}**

**}**

Examples of using Lambda Expression.

### Example of Custom Delegate



1. Public class Program
2. {
3. Public delegate void EmailSentNotificationDelegate(**string** notificationMsg);
4. Public static void Main(**string**[] args)
5. {
6. SendMailtoAllGoldMembers(WriteEmailSentNotificationOnConsole);
7. //SendMailtoAllGoldMembers(WriteEmailSentNotificationinLogFile);
8. //SendMailtoAllGoldMembers(WriteEmailSentNotificationinDatabase);
9. }
11. Public static void SendMailtoAllGoldMembers(EmailSentNotificationDelegate EmailSentNotification)
12. {
13. **foreach**(var emailId **in** GetListofEmailIdForGoldMembers())
14. {
15. //logic for sending the mail to all the gold members
16. System.Threading.Thread.Sleep(2000); //assuming that it will take 2 seconds to send a mail.
17. EmailSentNotification($ "Email Id {emailId} : mail sent");
19. }
20. }
22. Private static void InsertTheLogDetailsIntoDatabase(**string** msg)
23. {
24. //logic to insert notification in database table
25. }
27. Public static List < **string** > GetListofEmailIdForGoldMembers()
28. {
29. List < **string** > ListOfEmailIds = newList < **string** > ();
30. **for** (**int** i = 1; i <= 100; i++) {
31. ListOfEmailIds.Add($ "member{i.ToString("
32. D4 ")}@gmail.com");
33. }
35. **return** ListOfEmailIds;
36. }
38. Public static void WriteEmailSentNotificationOnConsole(**string** notificationMsg)
40. Console.WriteLine(notificationMsg);
41. }

44. Public static void WriteEmailSentNotificationinLogFile(**string** notificationMsg)
45. {
46. System.IO.File.AppendAllText(@ "D:\\delexample\log.txt", $ "\n {notificationMsg}");
47. }
48. Public static void WriteEmailSentNotificationinDatabase(**string** notificationMsg)
49. {
50. InsertTheLogDetailsIntoDatabase(notificationMsg);
51. }
53. }

## Events

### Event1:

s

## Collections

### Non-generic Collection:

Non-generic collections are under the namespace **System.Collections**

Stack,

Queue,

LinkedList,

SortedList,

ArrayList,

HashTable, etc.

### Generic Collection:

System.Collections.Generic

Stack<T>,

Queue<T>,

LinkedList<T>,

SortedList<T>,

List<T>,

Dictionary<TKey, Tvalue>

Note: Here the <T> refers to the type of values we want to store under them.

### Difference between IEnumerable and IQuerable:

<https://www.codeproject.com/Articles/732425/IEnumerable-Vs-IQueryable>

### Difference between HashTable and 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. |

## Static in C#

The static keyword can be applied to a class member including constructors, methods, properties, and events. Static can also be applied to classes.

The static modifier makes a member non-instantiable. The members of the class can be accessed via the class name (type) only.

**Static members in C#** - When are they loaded and how is the memory allocation

A class is loaded when either

• The static members of the class is accessed for the first time, or

• When the first instance of the class is created.

A class once loaded will remain in memory permanently and also are all static members of that class.

Series of events that occur when the first object is created,

1. Class is loaded.

2. Static members are loaded and allocated memory.

3. Static constructor is executed.

4. Object is created and Instance members are loaded and allocated memory.

5. Instance constructor is executed.

6. 4 and 5 repeats for subsequent object.

### Static class:

A class in C# can be declared as static and such a class can have only static members and instance members are not allowed.

**Static members in non-static class**

The instance variable can only access the non-static methods and variables, it cannot access the static methods and variables.

public class MyNonStaticClass

{

private static int myStaticVariable = 0;

public static void MyStaticMethod()

{

Console.WriteLine("This is static method.");

}

public void myNonStaticMethod()

{

Console.WriteLine("Non-static method");

}

}

**Static class in multiuser environment:**

Does static variables retain their values across user sessions?

Yes

### Stack Vs Heap:

When an instance of a reference type is created (usually involving the new keyword), only an object reference is stored on stack. The actual instance itself is created on the heap, and its’ address held on the stack.

Consider the following code:

1 void Method1()

2 {

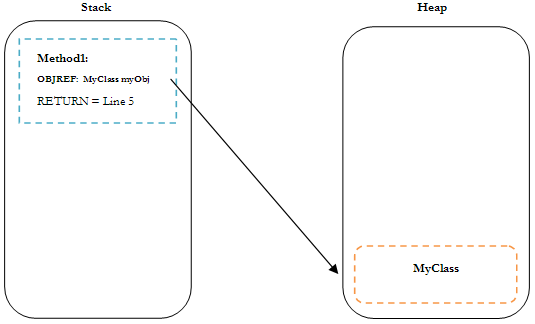
3 MyClass myObj=new MyClass();

4 Console.WriteLine(myObj.Text);

5 }

Listing 2: Code example using a reference type

In Listing 2 . a new instance of the class MyClass is created within the Method1 call.

****

<https://www.c-sharpcorner.com/UploadFile/de41d6/delegate-uses-in-C-Sharp/>

<https://www.c-sharpcorner.com/UploadFile/efa3cf/understanding-delegates-in-C-Sharp/>

----------------------------------------------------------------------------------------------------------------------------------

## Cross-Site Scripting Attack

**Cross**-**Site Scripting** (**XSS**) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web **sites**. **XSS** attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side **script**, to a different end user.

## Exception Handling in MVC

<https://www.c-sharpcorner.com/UploadFile/mscratnesh/exception-handling-in-mvc/>

<https://www.c-sharpcorner.com/UploadFile/ff2f08/exception-or-error-handling-in-Asp-Net-mvc-using-handleerror/>

## SOLID Principles

**SOLID Introduction**

1. SOLID principles are the design principles that help us to manage most of the software design problems
2. SOLID principle are used to make software designs more understandable, flexible and maintainable

**SOLID Acronym**

* **S** : **S**ingle Responsibility Principle (SRP)
* **O** : **O**pen closed Principle (OSP)
* **L** :  **L**iskov substitution Principle (LSP)
* **I**  :  **I**nterface Segregation Principle (ISP)
* **D** : **D**ependency Inversion Principle (DIP)

### Single Responsibility Principle

* "A class should have only one reason to change”
* Every module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class

### Liskov Substitution Principle

* Derived types must be changeable for their base types
* If a program module is using a Base class, then the reference to the Base class can be replaced with a Derived class without affecting the functionality of the program module

### Open/Closed Principle

* “Software entities should be open for extension, but closed for modification”
* The design and writing of the code should be done in a way that new functionality should be added with minimum changes in the existing code
* The design should be done in a way to allow the adding of new functionality as new classes, keeping as much as possible existing code unchanged

### Interface Segregation Principle

* We should not enforce clients to implement interfaces that they don't use. Instead of creating one big interface we can break down it to smaller interfaces.

### Dependency Inversion Principle

* Abstractions should not depend on the details whereas the details should depend on abstractions
* High-level modules should not depend on low level modules

### If we don’t follow SOLID Principles we

* End up with tight or strong coupling of the code with many other modules/applications
* Tight coupling causes time to implement any new requirement, features or any bug fixes and some times it creates unknown issues
* End up with a code which is not testable
* End up with duplication of code
* End up creating new bugs by fixing another bug
* End up with many unknown issues in the application development cycle

### Following SOLID Principles helps us to

* Achieve reduction in complexity of code
* Increase readability, extensibility and maintenance
* Reduce error and implement Reusability
* Achieve Better Testability
* Reduce tight coupling

### Solution to develop a successful application depends on

* **Architecture** : choosing an architecture is the first step in designing application based on the requirements. **Example :** MVC, WEBAPI, MVVM..etc
* **Design Principles** : Application development process need to follow the design principles
* **Design Patterns** : We need to choose correct design patterns to build the software

## Method Overriding Vs Method Hidding

https://www.geeksforgeeks.org/difference-between-method-overriding-and-method-hiding-in-c-sharp/

|  |  |
| --- | --- |
| **METHOD OVERRIDING** | **METHOD HIDING** |
| In method overriding, you need to define the method of a parent class as a virtual method using virtual keyword and the method of child class as an overridden method using override keyword. | In method hiding, you just simply create a method in a parent class and in child class you need to define that method using new keyword. |
| It only redefines the implementation of the method. | In method hiding, you can completely redefine the method. |
| Here overriding is an object type. | Here hiding is a reference type. |
| If you do not use override keyword, then the compiler will not override the method. Instead of the overriding compiler will hide the method. | If you do not use the new keyword, then the compiler will automatically hide the method of the base class. |
| In method overriding, when base class reference variable pointing to the object of the derived class, then it will call the overridden method in the derived class. | In the method hiding, when base class reference variable pointing to the object of the derived class, then it will call the hidden method in the base class. |

**-----------------------------------------------------------------------**

## Interviews Questions

SPA:

<https://msdn.microsoft.com/en-us/magazine/dn463786.aspx>

WEP API Security

<https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/>

**Custom routes call from UI:**

**<a href=”@Url.RouteUrl(“mymenu”)“>Main menu</a>**

Intranet: An intranet is a private network accessible only to an organization's staff.

An intranet application is a software data application used primarily on the internal network of an organization.

1. What is action method?
2. Can we write normal method in controller without using NoAction attribute? What is diff between Noaction and normal method ? what is use of NoAction.
3. Can we inherites private method of base controller into derived controller?
4. What are filters in mvc? Give the example of action filter. How can create a custom action filters in mvc?
5. Javscript: Select and de-select menu items using jquery? When select then apply .active class into menus e.g. Home,About Us, ContactUs menu items
6. How to insert 2 lacs records in users table in sql server? What are the ways to insert records into users tables
7. In Users table suppose if we wrong values are inserted into columns then how to correct them? Suppose FName contains LName and LName contains FName. Write query to update the records
8. Master DB is mandatory db can we delete it? Suppose if we create a custom DB Name “EmpDB” then is there some relations between master db and EmpDB?
9. Temp Variable? why we use it?
10. Inside sql function can we use exsiting custom table. Can we use insert statement inside function?

Filters in MVC:

<https://www.tutorialsteacher.com/mvc/filters-in-asp.net-mvc>

Difference b/w LocalStorage, sessionStorage and Cookies  
<https://scotch.io/@PratyushB/local-storage-vs-session-storage-vs-cookie>

<https://javascript.info/localstorage>

1. Create a sorting program in C#? Like bubble sorting etc.

<https://www.w3resource.com/csharp-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-3.php>

1. Arrays related questions
2. What is agile methodology? Scrum master
3. Print pyramid

<https://www.w3resource.com/csharp-exercises/for-loop/csharp-for-loop-exercise-20.php>

1. Print Fibonacci series:
2. Why static is introduced in C#. What is the main objective to use static class?
3. Why we use static void main method used in c#?
4. Why we use delegates? What is the main objective to use delegates?
5. Why we use extension method? We can also extend functionalities by other ways?
6. When to use Interface and Abstract class
7. Difference between Method overriding and Method hiding
8. Which design pattern you used in your application
9. What is EDMX in entity framework? What edmx file used in entity framework
10. ADO.NET
11. Architecture of project
12. Difference between ng-if and ng-show
13. Which access modifiers we can use in non-abstract method inside abstract class
14. Interface Vs Abstract class
15. Use of private constructor and copy constructors
16. Difference between Managed code and Unmanaged code
17. How to handle managed and unmanaged code
18. GC stage 0,1 2. GC release memory in every stage? How GC decide stage 0,1,2?
19. What authentication mechanism you used in WebAPI?
20. How OWIN use internally Oauth. Can we use another approach without oAuth? Can we implement OWIN usi
21. What is thread safe in c#?
22. What is the use of private constructor in singleton design pattern?