.Net Developer Quest

https://www.tutorialspoint.com/asp.net/asp.net\_validators.htm

* RequiredFieldValidator
* RangeValidator
* CompareValidator
* RegularExpressionValidator
* CustomValidator
* ValidationSummary

Client Side vs Server Side Validation

Be aware that any JavaScript input validation performed on the client can be bypassed by an attacker that disables JavaScript or uses a Web Proxy. Ensure that any input validation performed on the client is also performed on the server.

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

<https://stackoverflow.com/questions/7977305/default-value-for-asp-net-textbox-textmode-password>

As far as I know this is not default feature of text box. You should set the value of your password property using javascript when document is ready and then bind to focus event to remove it when someone wants to enter their password.

When using jQuery:

<asp:TextBox ID="headertxtPassWord" runat="server" Text="password" CssClass="header-login-input" ValidationGroup="A" TextMode="Password"></asp:TextBox>

<script type="text/javascript">

$(document).ready(function () {

// executes on document ready

$("#<%= headertxtPassWord.ClientID %>").val("password");

});

$("#<%= headertxtPassWord.ClientId %>").focus(function() {

$("#<%= headertxtPassWord.ClientId %>").val("");

});

</script>

<https://www.c-sharpcorner.com/interview-question/what-is-the-difference-between-dll-and-exe>

* exe 1).EXE is Executable File 2).exe is run individually 3).exe Has Main Function 4)Mainly is for standared alone application .DLL 1)DLL is Dynamic Link Library 2)dll can't run individually 3)dll doesn't contain Main Function 4)dll give support to ex

Exe assemblies are known as in-process components which were capable of running on their own as well as provide the support for others to execute. When we work with project templates like Windows Forms Applications, Console Applications, WPF Applications and Windows Services they generate an exe assembly when compiled.Dll assemblies are known as out-process components which were not capable of running on their own they can only support others to execute. When we work with project templates like Class Library and Windows Forms Control Library they generate a dll assembly when compiled.

.Exe 1.These are outbound file. 2.Only one .exe file exists per application. 3. .Exe cannot be shared with other applications. 4.exe is a executable file..dll 1.These are inbund file . 2.Many .dll files may exists in one application. 3. .dll can be shared with other applications. 4. dll is a Dynamic Link Library

<https://stackoverflow.com/questions/1210873/difference-between-dll-and-exe>

EXE:

1. It's a executable file
2. When loading an executable, no export is called, but only the module entry point.
3. When a system launches new executable, a new process is created
4. The entry thread is called in context of main thread of that process.

DLL:

1. It's a Dynamic Link Library
2. There are multiple exported symbols.
3. The system loads a DLL into the context of an existing process.

Well, the major differences are:

**EXE**

1. An exe always runs in its own address space i.e., It is a separate process.
2. The purpose of an EXE is to launch a separate application of its own.

**DLL**

1. A dll always needs a host exe to run. i.e., it can never run in its own address space.
2. The purpose of a DLL is to have a collection of methods/classes which can be re-used from some other application.
3. DLL is Microsoft's implementation of a shared library.

The file format of DLL and exe is essentially the same. Windows recognizes the difference between DLL and EXE through PE Header in the file. For details of PE Header, [You can have a look at this Article on MSDN](http://msdn.microsoft.com/en-us/library/ms809762.aspx)

The main **difference between** a **session** and a **cookie** is that **session** data is stored on the server, whereas **cookies** store data **in the** visitor's browser. **Sessions**are more secure than **cookies** as it is stored in server.**Cookie** can be turn off from browser.Jun 14, 2011

<https://stackoverflow.com/questions/23509617/why-does-net-generate-two-web-config-files-in-an-mvc-asp-net-application>

3down vote

ASP.NET configuration is stored in web.config files (XML files).

These files can appear in many directories in an ASP.NET application. They help to configure application behaviour even before after deploying, based on the fact that you can edit them with notepad. Also they keep separated your code and your configuration data.

Every web.config file applies to to the directory that it exists and ALL the child subdirectories. Web.config files in child directory may be used to override the parent's web.config file.

You have the option of overriding individual files or directories by using the location element. See [LINK](http://msdn.microsoft.com/en-us/library/b6x6shw7%28v=vs.90%29.aspx)

The settings inheritance rules are as foillows.

First there is the machine.config file which is located usually in systemroot\Microsoft.NET\Framework\versionNumber\CONFIG\

In the same directory exists a 'master' web.config file that defines settings for ALL asp.net application that are running in the machine.

Then come your web.config files that exist in your application.

More Info:

--

@@IDENTITY Is a system function that returns the last-inserted identity value

SCOPE\_IDENTITY

Returns the last identity value inserted into an identity column in the same scope. A scope is a module: a stored procedure, trigger, function, or batch. Therefore, if two statements are in the same stored procedure, function, or batch, they are in the same scope.

--

<http://blog.exsilio.com/all/pros-and-cons-of-dynamic-sql/>

## Dynamic SQL

Definition: Dynamic SQL is a batch of SQL statements that are generated within T-SQL and executed using the Execute (or exes) statement, or via sp\_executesql system stored procedure.

### **Benefits of Dynamic SQL**

**Predicate Optimization:**The real benefit of dynamic SQL is that the execution plans generated for each invocation of the query will be optimized for on the predicates that are actually being used at that moment.  The main issue with the static SQL solutions, aside from maintainability, was that the additional predicates confused the query optimizer, causing it to create inefficient plans.  Dynamic SQL gets around this issue by not including anything extra in the query.

**Single Query Plan Caching:** For every stored procedure there is one cached query plan and an additional ad hoc plan caches for each invocation of the stored procedure (this can be verified using the view sys.dm\_exec\_cached\_plans).   This means that every time a new argument is passed to the stored procedure, a compilation occurs, which is clearly going to kill performance.  The dynamic query is not being parameterized and is therefore producing duplicate query plans for different arguments.

### **Drawbacks of Dynamic SQL**

**Speed:** Dynamic SQL tends to be slower than static SQL, as SQL Server must generate an execution plan every time at runtime.

**Permissions**: Dynamic SQL requires the users to have direct access permissions on all accessed objects like tables and views.  Generally, users are given access to the stored procedures which reference that tables, but not directly on the tables.  In this case, dynamic SQL will not work.

**Syntax**: One distinct advantage of writing stored T-SQL procedures is that you get a syntax check directly.  With dynamic SQL, a trivial syntax error may not show up until run time.  Even if you test your code carefully, there may be some query, or some variation of a query, that is only run in odd cases and not covered in your test suite.

2) WHERE clause is used for filtering rows and it applies on each and every row, while **HAVING** clause is used to filter groups in **SQL**. 3) One syntax level difference between WHERE and **HAVING** clause is that, former is used before GROUP BY clause, while later is used after GROUP BY clause.

<https://www.c-sharpcorner.com/UploadFile/009464/bind-gridview-using-dataset-in-Asp-Net/>

<https://www.c-sharpcorner.com/UploadFile/009464/bind-gridview-using-dataset-in-Asp-Net/>

the Web Form.

<form id="form1" runat="server">

<div>

<asp:GridView ID="gvitems" runat="server" AutoGenerateColumns="false">

<Columns>

<asp:BoundField DataField="ItemId" HeaderText="Num" ItemStyle-Width="30" />

<asp:BoundField DataField="ItemName" HeaderText="Country" ItemStyle-Width="150" />

</Columns>

</asp:GridView>

</div>

</form>

Code Behind

using System.Data;

if (!Page.IsPostBack)

{

DataTable dt = new DataTable();

dt.Columns.AddRange(new DataColumn[2] { new DataColumn("Num", typeof(int)),

new DataColumn("Country",typeof(string)) });

dt.Rows.Add(1, "USA");

dt.Rows.Add(2, "France");

dt.Rows.Add(3, "Spain");

gvitems.DataSource = dt;

gvitems.DataBind();

}

<https://www.c-sharpcorner.com/UploadFile/francissvk/gridview-highlighting-row-or-cell/>

protected void gvitems \_RowDataBound(object sender, GridViewRowEventArgs e)

{

if (e.Row.RowType == DataControlRowType.DataRow)

{

string country = Convert.ToString(e.Row.Cells[2].Text.ToString());

if (country == ‘USA’)

{

e.Row.BackColor = Color.Red;

}

}

}

<https://stackoverflow.com/questions/15743192/check-if-number-is-prime-number>

var number;

Console.WriteLine("Accept number:");

number = Convert.ToInt32(Console.ReadLine());

if(IsPrime(number))

{

Console.WriteLine("It is prime");

}

else

{

Console.WriteLine("It is not prime");

}

public static bool IsPrime(int number)

{

if (number <= 1) return false;

if (number == 2) return true;

if (number % 2 == 0) return false;

var boundary = (int)Math.Floor(Math.Sqrt(number));

for (int i = 3; i <= boundary; i+=2)

{

if (number % i == 0) return false;

}

return true;

}