**ASP.NET**

ASP.Net의 개념 및 특징, 작동 원리

이벤트 처리

Html 컨트롤

웹 컨트롤

ASP.NET 페이지의 요소

Page 클랫의 IsPostBack 프로퍼티

페이지 실행 주기

숨겨진 ViewState

비주얼 스튜디오 닷넷을 이용한 ASP.NET

**[데스크 탑과 웹 응용 프로그램의 차이 ]**

1. 항상 네트웤으로 연결되어 있슴

HTTP : 항상 2개이상의 네트웤으로 연결된 머신이 필요

- Hypertext Transfer Protocol

2. 상태가 없는 와이어 프로토콜임

- 웹 서버는 클라이언트 브라우저에 응답을 보내자 마자 이전의 상호작용에 대해서는 모두 잊어버림

- 따라서 클라이언트에 대한 정보 및 상태를 관리하기 위한 코드가 추가됨(세션 변수, 쿠키, 응용

프로그램 변수, 캐쉬)

\*) 웹 응용 프로그램 개발시 필요한 개념

- 기초 개념

HTTP, HTML, 클라이언트 측 코드 베이스, 서버 측 코드 베이스

웹 서버(Microsoft IIS)의 역할 이해 필요

- ASP.NET의 페이지 구성과 ASP.NET 웹 컨트롤

기존의 COM기반 ASP보다 우수

프리젠테이션 로직과 비즈니스로직 별도 파일 구분 가능

스크립트 언어가 아닌 프로그래밍 언어로 구현 가능

**[ 웹 응용 프로그램의 일반적 시나리오 ]**

1. 클라이언트 머신이 웹 브라우저를 실행하면,

서버 머신에 있는 특정 리소스(\*.aspx 파일이나, \*htm 파일)에 접근하기 위해 HTTP 요청이 만들어짐

- HTTP : 표준 요청/응답 패러다임 위에 만들어진 텍스트 기반의 프로토콜

ex) http://www.daum.net

브라우저 소프트웨어가 도메인 네임(DNS) 기술을 이용해 등록된 URL을 IP로 변환

브라우저는 소켓 연결을 열고(일반적으로 80포트)

HTTP요청을 해당 웹 사이트에 있는 기본 페이지로 보낸다.

2. 호스트하는 웹 서버가 HTTP 요청을 받게 되면,

- 지정된 리소스에서 클라이언트가 제공한 입력값들을 추출해서 적당한 HTTP 응답을 만든다.

웹 개발자는 여러가지 기술(CGI, ASP, ASP.NET, Java Serlerts등)을 이용해서 HTTP 응답으로 보내

질 내용을 동적으로 생성

3. 클라이언트 측 브라우저가 웹 서버가 보낸 HTML 을 그려내게 됨

**[ 웹 응용 프로그램과 웹 서버 ]**

1. 웹 응용 프로그램 : 웹 서버의 특정 디렉토리에 저장되어 있는 파일들

(\*.htm, \*asp, \*aspx, 이미지 파일)과

관련 컴포넌트(.NET or COM바이너리)의 집합임

2. 웹 서버 : 웹 응용 프로그램을 호스트하는 소프트웨어 제품

통합 보안, 파일 전송 프로토콜(FTP)지원, 메일 익스체인지 서비스등 여러 가지 관련 서비스를 제공

\*) 인터넷 정보 서비스(IIS) : Microsoft의 엔터프라이즈 급 웹 서버 제품

ASP와 ASP.NET 웹 응용 프로그램을 지원함

\*) 즉, ASP.NET 웹 응용 프로그램을 구현하려면, IIS를 사용해야 함

1) IIS

인터넷 정보 서비스 : IIS 관리 유틸

IIS 가상 디렉토리 :

하나의 IIS는 각 가상 디렉토리에 있는 여러개의 웹 응용 프로그램을 호스트 할 수 있슴

가상 디렉토리는 로컬 하드 드라이브의 물리적인 디렉토리에 매핑

ex) Test 라는 가상 디렉토리를 하드 드라이브에 생성

외부 접근시 : http://www.Test.com으로 접근가능

IP가 등록이 안되어 있다면 IP주소 포함해서 접근됨

http://www.127.0.0.1:8080/Test

생성방법

- 수작업

- .NET 자동

2) HTML

- 가상 디렉토리에 파일들을 구성시켜야 함

웹 응용 프로그램이란 어떤 사이트의 기능을 이루는 일련의 파일들임

이 파일들의 대부분은 HTML(Hypertext Markup Language)에 의해 정의된 문법적 토큰을 표함하

고 있다.

- HTML은

리터럴 텍스트, 이미지, 외부링크 및 다양한 HTML기반의 GUI위젯들이 클라이언트 측 브라우저에

서 어떻게 그려질지를 설명하는 데 이용되는 표준 마크업 언어임

**[ 웹 프로그래밍 ]**

- 서버-사이드 프로그래밍(서버에서 실행되는 프로그램을 작성하는 것을 의미)의 한표현

사용자가 클라이언트 어플리케이션을 통해서 서버에 무엇인가를 요청한다면, 서버에서는 요청한 것을

수행하고 결과를 다시 클라이언트에게 보낸다.

여기서 서버 측에서 돌아가고 있는 프로그램을 작성하는 것을 말함

- 웹 프로그래밍

클라이언트와 HTTP 프로토콜을 통해서 통신하고 HTML이나 XML 같은 형식으로 데이터를 교환할 때,

서버에서 동적인 페이지를 만들어 내는 프로그래밍을 의미

웹 프로그래밍은 브라우저와 같은 클라이언트의 요구를 동적으로 처리하고, 그 결과를 다시 HTML이

나 XML 형식으로 클라이언트에게 전달함

- 닷넷에서는 윈폼프로그램과 웹 프로그램을 구별하지 않음

모두 닷넷 기본 클래스 라이브러리를 사용함

개발 방법도 유사

두 개의 프로그래밍을 위해 별도의 프로그래밍 언어를 사용하지 않음

C#을 통해 두 개의 프로그래밍 가능

비주얼 스튜디오 닷넷을 사용하면 거의 동일한 방식으로 프로그래밍을 할 수 있게 됨

**[ ASP.NET이란 ]**

- 웹 서버의 서버 사이드 개발 플렛폼

ASP는 HTTP와 HTML과 같은 표준을 기반으로 하는 서버-사이드 어플리케이션을 구축하기 위한 프레

임워크임

ASP.NET은 더 나아가서 XML과 SOAP과 같은 표준들도 적극적으로 수용하고 있슴

- 언어가 아니다.

ASP.NET에서는 CLR을 따르는 모든 언어로 개발가능, 특정언어로 개발을 해야 되는 것은 아님

- MS사에서 개발한 웹 서버용 스크립트 환경

ASP의 발전된 형태

과거 Framework SDK안에서 ASP+이라는 이름으로 발전되다가 현재 ASP.NET이라는 이름으로 발전됨

**[ ASP와 ASP.NET의 차이 ]**

0 )ASP : VBScript나 JScript 스크립트 언어만 사용가능

스크립트 언어는 인터프리터 방식이기 때문에 컴파일 방식보다 느림

ASP.NET : 동작하는 모든 프로그래밍 언어를 사용할 수 있다.

모두 컴파일 방식이기 때문에 속도가 빠르다.

C#의 강력함 + .NET 기본 클래스 라이브러리를 사용한 웹 개발....

ASP와 비교할 수 없는 생산성을 가지게 됨

1) 기존의 ASP의 모든 객체들을 그대로 사용

- Request, Response, Sesstion, Application, Server 객체 그대로 사용

- 기존 ASP 소스를 ASP.NET용으로 포팅하지 않아도 기존의 웹 어플리케이션들은 돌아감

2) 한번 수행할 때 컴파일 된다.

- 처음 수행될 때 컴파일 되므로 컴파일러 언어와 마찬가지로 빠른 속도의 실행가능

(중간언어의 형태로 컴파일 됨)

- 개발자가 ASPX의 내용을 바꾸어 서버를 업데이트 시켰다면, 바뀐 파일의 첫번째 억세스시 새롭게

컴파일 되어 서비스 적용됨

기존 ASP는 예전버전이 그대로 번역되어 실행되는 구조였슴

3) 새로운 .NET 플랫폼에 맞춰 기존의 DLL Hell라고 불리던 컴포넌트 작성의 어려움이 해소됨

- ASP에서는 레지스트리나 시스템정보등을 알아내기 위해서 따로 VC or VB등을 사용하여 컴포넌트를 만들어야 했슴

- ASP.NET에서는 BCL을 호출하는 것이 가능하기때문에 그리고 ASP.NET자체도 컴파일이 가능하기 때문에 ASP.NET 자체만으로도 서버 시스템의 모든 것에 억세스가 가능

4) 서버 사이드 이벤트 발생

- 사용자가 웹 브라우저에서 어떤 행동을 취할 경우, 이 행동들이 서버에 전달되어 적당한 처리 가능

5) 컨트롤들이 더욱 풍부해짐

- 유효성 검사 컨트롤, Calendar 컨트롤등 추가

유효성 검사를 하기 위해 자바 스크립트 소스나 각종 컴포넌트를 만들지 않아도 됨

6) Cache 의 사용으로 더욱 더 빠른 처리 가능

- Data Cache등 각종 Cache의 사용으로, DB에 사용자가 접속할때마다 DB 내용을 억세스할 필요가 없

어짐

7) 로직의 분리 가능

- 웹 폼을 통한 화면 디자인 가능

- ASPX에서는 Business Logic을 구현함으로 로직 분리 가능

ASP : HTML코드와 스크립트 언어를 사용한 코드가 같은 .asp 파일에 존재

ASP.NET : 디자인을 위한 HTML코드와 로직을 위한 코드를 완전히 분리한다.

가독성, 디자이너와 개발자간의 분리 작업 가능

=> 코드-비하인드(code\_behind)방식이라 함

8) ASP는 디버깅 방법이 별도로 존재하지 않음

대부분의 ASP개발자들은 코드 사이에 Response.Write 메서드를 호출해서 디버깅을 함

ASP.NET : 웹 어플리케이션하에서의 디버깅 지원

9) ASP : 클라이언트의 종류에 따라 코드 작성

ASP.NET : 하나의 코드 페이지를 요청한 클라이언트에 맞게 자동으로 설정해서 보냄

웹 폼이라는 컨트롤을 통해 가능케 함

10) 웹 프로그래밍도 객체 지향 프로그래밍임

하지만 기존의 웹은 절차 지항적 프로그래밍에 가까움

ASP.NET은 완벽한 객체지향 프로그래밍 개발 방법임

모든 페이지는 System.Web.UI.Page 클래스를 상속하기 때문에,

Page 클래스에 들어있는 모든 기능을 재사용할 수 있다.

|  |  |  |
| --- | --- | --- |
|  | **ASP** | **ASP.NET** |
| Client | Html, VBScrip, JSScript | Html, VBScrip, JSScript |
| Server | VBScrip,  Script,  ADO | WebForm  .NET Language(VB, C#)  ADO.NET |

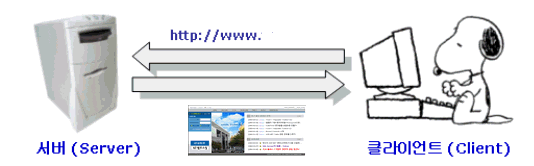
**[ ASP.NET을 사용하기 위한 환경 ]**

- 윈도우 200 프로페셔널, 윈도우 2000 서버, 윈도우 XP 프로페셔널

- IIS 웹 서버

- 닷넷 프레임워크 SDK

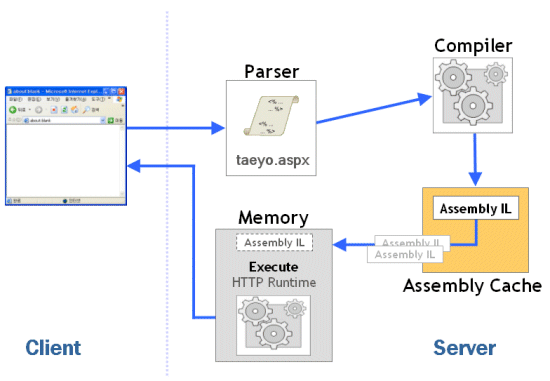
**[ ASP.NET 작동 원리 ]**



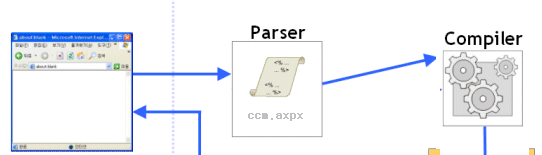
\*) URL에 해당하는 ASP File을 처리한 뒤 HTML형태로 바꾸어서 전송

1. 가장 처음에 클라이언트는 IIS에게 ASP 파일을 요구
2. IIS는 해당 ASP페이지를 처리하기 위해서 ASP.DLL이란 DLL을 로딩하고 이 DLL안에서 해당 ASP페이지를 처리하여 결과를 클라이언트에게 HTML로 리턴
3. 클라이언트 웹 브라우저는 이 HTML을 받아서 화면에 렌더링 수행

**[ ASP.NET 실행 구조1 ]**



**[ ASP.NET 실행 구조2 ]**



1) 클라이언트는 서버로 특정 aspx (ASP.NET 페이지)를 호출

2) ccm.aspx 라는 페이지를 달라는 클라이언트의 호출을 받은 웹 서버는 이 파일을 찾고, 이 파일의 확장

자를 확인

2.1) 만일, 이 파일의 확장자가 htm 이라면 서버는 찾은 파일을 클라이언트에게 넘겨준다.

2.2) 파일의 확장자가 aspx 라는 것을 확인하면,

서버는 이 확장자와 연결되어져 있는 DLL (ASP.NET)에게 이 파일의 처리를 의뢰

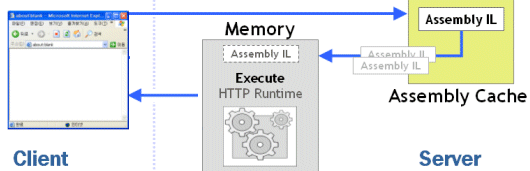
3) [Parser] ASP.NET은 이 파일을 받아서 일단 파싱하는 작업을 수행함

- 파서는 .aspx파일의 내용을 해석하고 올바른지 검사

- 올바르다면 컴파일러에게 이 파일을 전달함, C#코드로 작성하였으면 C#컴파일러에게 전달됨

4) [컴파일러] 라인 단위의 파싱작업을 끝낸 다음에 이 파일은 서버에 의해서 일단 컴파일됨

- 중간 언어로 컴파일함



6) 컴파일이 되어지면 MSIL과 MetaData로 만들어지고, 관리 코드인 Assembly IL 생성(확장자:DLL or EXE)

7) [어셈블리캐시]관리 코드로 만들어진 것들은 어셈블리 캐쉬라는 공간에 쌓여지게 됨(ASP.NET 특성)

- 일단 컴파일되어진 aspx 페이지들은 그 중간완성품을 캐쉬에 저장해 놓고,

이후 여러 사용자들이 같은 aspx 페이지를 요청할 경우, 캐쉬에 이미 존재하는 IL을 재사용 이로

인해 상당한 속도의 증가효과를 얻어낼 수 있슴

8) [메모리]어셈블리 캐쉬에 올라와 있는 IL 들은 요청에 대한 응답을 기다리고 있는 사용자들에게 결과물

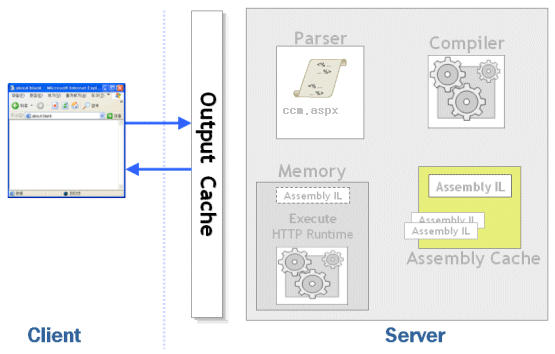
을 건네 주어야 하기에 실행을 실시

- 캐쉬에 있던 IL 은 메모리로 로딩, 이것은 CLR의 Jitter 에 의해 다시 한번 컴파일됨(네이티브코드)

- 그 결과 만들어지는 결과물들(대부분의 경우 동적으로 생성된 HTML)을 클라이언트의 브라우저에게

넘겨주게 됨

**[ ASP.NET 실행 구조3 ]**



\*) 출력 캐쉬 : 사용자에게 최종으로 넘겨줄 두번째 컴파일 버전의 결과물을 캐쉬에 저장해 두는 방법

Output Cache 라는 특별한 캐쉬구역에 저장해 두게 되면,

사용자의 요청은 파싱, 코드 컴파일 단계, 런타임 컴파일 단계등을 거치지 않고,

Output Cache 구역에 존재하고 있는 결과물로써 대단히 빠르게 응답해 줄 수 있게 됨

**[ 정리 ]**

**ASP & ASP.NET**

**ASP**

- .asp 확장자를 쓰는 페이지

- 사용할 수 있는 서버 스크립트 VBScript와 JScript

대부분의 개발자는 VBScript를 사용

- HTML과 ASP 코드가 섞여있는 구조

- 사용자의 요청시 매번 해석되고, 그 결과 만들어지는 HTML이 사용자에게 보내어지는 흐름 구조

**ASP.NET**

- .aspx 라는 확장자(ASP+ 에서 유래 : +를 옆으로 약간 돌린 x 가 사용)

- 컴파일 언어인 VB.NET, C#등의 .NET 언어가 사용) 강력한 데이터 타입을 제공

스크립트로써는 제한적이었던 기능을 Base Class Library를 통해서 제공

컴파일 되고 메모리에 올라서서 실행되어지기에 스크립트에 비해서 월등한 성능의 향상

어떠한 .NET 언어로 프로그래밍을 하던 모두 같은 결과를 얻어낼 수 있슴

VB.NET 언어로 제작한 클래스를 C#으로 상속받아 확장할 수 있는 등의 언어 교차 상속을 가능

* ASP.NET은 ASP 때에는 없었던 강력한 서버 컨트롤들을 제공

드래그 앤 드롭만으로 컨트롤을 가져다가 쓸 수 있었던 것처럼, 비슷한 방식으로 쉽게 사용할 수 있

는 수 많은 서버 컨트롤들이 제공

\*) ASP 때는 Input 컨트롤등의 제한적인 Html 컨트롤만을 개발자가 태그로 작성하여 사용할 수 밖에 없었슴

- ASP.NET에서는 코드와 컨텐트를 분리해서 2개의 파일로 작성하여 사용할 수 있는 코드 비하인드 제공

두 페이지는 컴파일시 하나로 묶여 유기적으로 동작

마치 하나의 어셈블리처럼 구성되어져 동작하게 됨

이러한 방법은 페이지의 컨텐트와 소스 코드를 분리해서, 페이지의 관리, 유지 보수를 용이하게 하며, 페이지의 확장, 상속을 가능하게 해 줌

**ASP와 ASP.NET은 서버측의 기술임 (이것은 클라이언트 측과는 전혀 상관없음을 의미함)**

ASP.NET은 ASP(Active Server Pages) 라는 기술에 .NET 기술을 접목한 기술임

그 페이지의 결과물은 클라이언트를 위한 것이라 하더라도, 그 결과물을 만들어 내는 곳은 서버측임

(결과물은 대부분 화면을 꾸미기 위한 HTML 에 불과)

여전히 클라이언트 스크립트를 사용해야 함

기존처럼 자바스크립트나 VBScript를 여전히 클라이언트와 대화하는 도구로 사용함

따라서 클라이언트 스크립트도 습득해야 할 내용임

또한, 동적이면서도 화려한 웹 페이지를 클라이언트에게 보여주고 싶다면 DHTML 도 필요함

**ASP.NET에서 제공하는 새로운 기술**

- 웹 폼(Web Forms)

: ASP.NET의 새로운 프로그래밍 모델이며, ASP의 장점을 Visual Studio.NET의 개발 편의성, 생산성과

결합한 모델

- 서버 컨트롤(Server Controls)

: 웹 폼 프로그래밍 모델의 주요 구성 요소이며, HTML 요소들과 대략적으로 매핑되어짐

- 웹 서비스(Web Services)

: 프로그래밍 가능한 서비스의 제작을 이루는 ASP.NET 의 핵심 기술. 웹 서비스라는 기술은

Microsoft의 .NET 환경에서만 동작하는 기술이 아니라 XML을 지원하는 모든 플랫폼에서 제공,

사용 가능한 기술이기에 앞으로의 세상은 웹 서비스의 세상일 것이라는 기대는 실제화될 것으로 보임

- 캐싱(Caching)

: ASP.NET은 강력한 캐싱 엔진을 탑재하고 있다. 이를 통해 웹 페이지의 캐싱을 효과적으로 구성

할 수 있으며, 효과적인 캐싱을 통해 어플리케이션의 성능 향상을 가져올 수 있다. 또한, 효과적

으로 구성한 캐싱은 서버간의 프로세싱 로드를 절감하는 효과를 가져오기에, 전체적인 네트웍의

트래픽 또한 향상되는 효과를 얻게한다.

- 향상된 상태 관리(State management Improvements)

: 웹 서버상에서 세션 상태를 분산시켜 저장할 수 있도록 하는 기능을 제공 별도의 상태 서버

나 SQL Server Database를 통해 상태 정보 유지하도록 할 수 있으며, 클라이언트 브라우저가

쿠키를 지원하지 않아도 세션의 상태를 관리할 수 있는 방법 또한 제공

**[ C#으로 ASP.NET 페이지 만들기 ]**

예제1) 간단한 ASP.NET 페이지 (hello.aspx)

<%@ Page language= "C#" %>

<%

this.Response.Write("Hello ASP.NET");

%>

🡺 [인터넷 서비스 관리자] 가상 디렉토리 생성 후 실행 : hello

( 읽기및 스크립트실행)

<http://localhost/hello/hello.aspx>

🡺 웹 폴더로 만들고 실행하는 방법

ASP.NET 페이지 분석

<%@ Page language= "C#" %>

- language : 이 페이지에서 사용하는 언어(없으면 VB.NET으로 인식함)

<%

this.Response.Write("Hello ASP.NET");

%>

- <% %> 사이에는 C#으로 작성된 코드가 들어간다.

- 클라이언트에게 문자열을 전송해주는 코드임 윈폼과 동일

클라이언트 요청 🡪 파서(Parser) 🡪 컴파일러 🡪 어셈블리캐시(Assembly IL)

(hello.aspx) hello.exe 임시저장

🡪 메모리(Assembly IL) 🡪 출력 캐시 🡪 클라이언트에게 전송

실행

< hello.aspx가 HTMP코드로 변환되며 클라이언트에 전송되는 과정 >

[ hello.aspx 파일을 처음 호출했을때의 시나리오 재확인 ]

- 브라우저와 같은 클라이언트에서 hello.aspx 파일 요청

- 해당 파일을 소유한 웹서버는 출력 캐시를 검색해서 캐시되어 있는 결과

페이지 존재 유무 파악

- 요청한 hello.aspx페이지는 파서에 의해서 내용이 해석됨

- 이 때 .aspx페이지는 Page 클래스를 상속받는 완전한 타입 형태로 변환됨

예)

using System;

public class hello\_aspx : System.Web.UI.Page

{

Response.Write("hello.NET");

}

\*\*) ASP.NET 페이지는 모두 System.Web.UI.Page 클래스를 상속받음

- 이렇게 해석된 hello.aspx 페이지는 중간 언어로 컴파일되기 위해서 C# 컴파일러에게 전달됨

- 에러가 없다면 중간 언어로 되어 있는 .dll 파일을 생성할 것임

나중의 성능을 위해 어셈블리 캐시에 저장됨

- 이 중간코드는 다시 네이티브 코드로 컴파일되어서 웹 서버에서 수행됨

- 네이티브 코드는 클라이언트에게 전달해 줄 응답 페이지를 생성함

이것은 출력 캐시에 저장됨

존재 확인 ) 캐시 되어 있는 페이지들이 폴더 별로 저장되어 있슴

C:\\Windows\Microsoft.NET\Framework\V1.3705\Temporary ASP.NETFiles\

해당 파일의 dll을 ILDASM도구로 확인해 보자.(중간 코드)

이 중간 코드는 다시 네이티브 코드로 컴파일되어서 웹 서버에서 수행됨

- aspnet\_wp.exe 포로세스에서 수행됨(작업관리자 확인)

네이티브코드는 클라이언트에게 전달해 줄 응답페이지를 생성함, 그리고 이것은 출력캐시에 저장되

게 됨

예제2) 메서드 추가하기

<%@ Page language= "C#" %>

<script runat=server>

string str;

public void Show(string s)

{

this.Response.Write(s);

}

</script>

<%

str = "안녕하세요 ASP.NET입니다.";

Show(str);

%>

- 타입의 정의와 사용이 구별되어 있다.

- ASP.NET 페이지는 하나의 클래스 이다.

str : 클래스의 필드

show() : 클래스의 메서드

- <script runat=server> ~~ </script>

사이에는 생성될 클래스의 정의 부분에 추가되는 것임

- <% ~~ %>

사이에는 생성될 클래스의 객체를 암시적으로 사용함

\*) runat="server"

이 안에 있는 코드가 서버에서 수행된다는 것을 말함

예제3) HTML태그와 같이 사용하기

예제 2의 코드와 HTML코드를 같이 사용할 수 있슴

<%@ Page language= "C#" %>

<html>

<head></head>

<body>

<h1> 이것은 HTML 코드입니다.</h1>

<script runat=server>

string str;

public void Show(string s)

{

this.Response.Write(s);

}

</script>

<%

str = "안녕하세요 ASP.NET입니다.";

Show(str);

%>

<p>

<i> 이것도 HTML 코드입니다.</i>

</body>

</html>

- 타입의 정의와 사용이 구별되어 있다.

예제4) 클라이언트-사이드 스크립트와 같이 사용하기

클라이언트 자체에서 수행되는 스크립트를 말함

ex) 자바스크립트와 같은 것은 브라우저에서 해석하여 결과를 보는 것임

<script...> </script> 사이에 작성하면 됨

<%@ Page language= "C#" %>

<html>

<head></head>

<body>

<h1> 이것은 HTML 코드입니다.</h1>

<script runat=server>

string str;

public void Show(string s)

{

this.Response.Write(s);

}

</script>

<%

str = "이것은 서버에서 수행됩니다.";

Show(str);

%>

<p>

<i> 이것도 HTML 코드입니다.</i>

<script language = "JavaScript">

alert("이것은 클라이언트에서 수행됩니다.");

</script>

</body>

</html>

- 타입의 정의와 사용이 구별되어 있다.

- <script language = "JavaScript">

runat="server"가 명시되어 있지 않으므로, 이것은 클라이언트에서만 수행된다.

예제5) ASP.NET 코드 비하인드 사용하기

- 지금까지의 예제는 모두 ASP와 흡사한 방식임

- ASP의 단점은 디자인을 위한 HTML코드와 로직을 위한 C#코드가 섞여있슴

**//default.aspx.cs**

using System;

using System.Web.UI;

public class Hellopage : System.Web.UI.Page

{

public string str;

public void Show(string s)

{

Response.Write(s);

}

}

**// default.aspx**

<!-- default.aspx-->

<%@ Page language="C#" src="Default.aspx.cs" Inherits="HelloPage"%>

<html>

<head></head>

<body>

<h1> 이것은 HTML 코드입니다.</h1>

<%

str = "이것은 서버에서 수행됩니다.";

Show(str);

%>

</body>

</html>

- src : 상속하고자 하는 페이지 클래스가 있는 위치의미

- Inherits : 상속하는 클래스명 명시

\*\*) .cs 파일을 컴파일 하는 것이 아니라(직접 컴파일할 필요가 없다)

클라이언트가 default.aspx 페이지를 요청하게 되면

내부적으로 이 페이지가 상속하는 클래스가 있는 파일을 컴파일 한다.

예제6) cs 파일을 dll 파일로 변환

- 코드 비하인드의 또다른 장점

.cs 파일을 .dll파일로 만들수 있다.

어플리케이션 로직이 들어있는 .cs 파일을 노출시키지 않도록 할 수 있다.

명시적으로 .dll파일로 변환해 놓으면 중간 코드로 컴파일하는 과정을 미리하게 되므로 속도 향상에 도

움을 줌

csc /target:library hello5.aspx.cs

이 명령어를 수행하면 hello5.dll파일이 만들어짐

소스파일인 hello5.aspx.cs 파일을 삭제하고

hello5.aspx 파일이 위치한 곳에 "bin"이라는 폴더를 만들고

hello5.aspx.dll파일을 위치함

마지막으로 hello5.aspx 파일의 Page지시에서 Src 속성을 제거

**// default.aspx**

<!-- default.aspx-->

<%@ Page language="C#" Inherits="HelloPage"%>

\*) 웹 폼

- 표현 부분 ( 웹 폼의 시각적인 부분을 담당) : display

- 코드 부분 ( 웹 폼의 내부 로직 부분을 담당) : logic

\*) 웹 폼 표현 모델

- 단일 페이지 모델(Single-File Page Model)

장점 : 코드가 많지 않을 경우 코드와 태그를 한 곳에서 확인할 수 있으므로 쉡게 웹 폼을 분석

단일 페이지 모델로 작성한 웹 폼은 파일이 하나뿐이므로 다른 개발자에게 배포나 보내기가

쉬움

파일간의 종속성이 없으므로 단일 페이지의 이름을 쉽게 수정

- 코드 숨김 페이지 모델(Code-Behind Page Model) : .aspx파일(디스플레이) / .cs파일(로직)

장점 : 개발자가 코드를 작성하는 동안 웹 디자이너는 태그로 디자인 작업 진행

웹 디자이너에게 코드를 노출하지 않으므로 개발자와 웹 디자이너간의 혼선을 줄일 수 있슴

코드를 여러 페이지에서 재사용 할 수 있슴

**[ VisualStudio.NET 2005로 구현하기 ]**

ASP.NET 웹 사이트 선택

* 생성 파일 : Defaul.aspx : logic

Default.aspx.cs : display

* Default.aspx.cs 파일을 삭제하면?? 종속 관계인 cs 파일이 같이 삭제됨

실습1) 단일 페이지 모델의 웹 폼과 코드 숨김 페이지 모델의 웹 폼 구현

[새항목 추가] – Web Form 설정(Ex01.aspx)

* 생성 파일 : 하나의 aspx파일 생성됨

[새항목 추가] – Web Form 설정(Ex02.aspx) : 다른 파일에 코드 입력 버튼 클릭

* 생성 파일 : 하나의 aspx파일과 하나의 cs 파일이 생성
* 기본적으로 Load 이벤트처리기가 등록되어 있슴

[단일 파일 페이지 모델의 웹 폼에 Load 이벤트 처리기 등록하기]

Ex01.aspx파일을 더블클릭하여 소스뷰에 나타냄

상단의 ‘개체 및 이벤트’ 드롭다운 리스트에서 page개체를 선택

제공하는 이벤트 리스트들 중 Load 이벤트 선택

* 웹 폼의 코드 부분에 자동 등록됨
* protected void Page\_Load(object sender, EventArgs e)
* {
* Response.Write("Load 이벤트가 발생했습니다.");
* }

**[ ASP.NET으로 이벤트 프로그래밍하기 ]**

ASP.NET의 장점 중 하나

- 윈폼에서의 이벤트기반 프로그래밍 방식과 동일

- System.Web.UI.Control 클래스는 윈폼에서 사용했던

System.Windows.Forms.Control과 유사함

- Page 클래스의 이벤트 발생 순서

1) Page\_Init :

페이지가 처음 초기화될 때 발생

페이지가 실제 로드되기 전에 초기화를 담당

2) Page\_Load :

페이지가 로드 될때 발생

클라이언트가 브라우저를 가지고 페이지를 요청할 때마다 발생함

페이지에 있는 초기화된 컨트롤들의 속성을 설정한다.

3) Page\_Unload :

페이지가 더 이상 필요없을 때 발생

클라이언트가 브라우저를 통해서 또 다른 페이지로 이동했을 때 발생

사용한 파일이나 데이터베이스들과 같은 리소스들을 해지 할 때 사용

\*\*) Page\_Init ==> Page\_Load ==> 클라이언트의 요청처리 ==> Page\_Unload

< !- - event..aspx 페이지 -- >

<%@ Page language="C#" AutoEventWireup="false" src="event.aspx.cs" Inherits="EventPage"%>

<html>

<head></head>

<body>

<h1> 이것은 HTML 코드입니다.</h1>

</body>

</html>

\*) AutoEventWireup 속성 : true이면 Page클래스의 이벤트를 자동생성해줌

즉 이벤트 핸들러를 추가 하는 코드를 작성하지 않아도 됨

// event.aspx.cs

//.aspx.cs

using System;

using System.Web.UI;

public class EventPage : System.Web.UI.Page

{

public string str;

public void Show(string s)

{

Response.Write(s);

}

public EventPage()

{

// this.Init += new System.EventHandler(this.Page\_Init);

}

void Page\_Init(object sender, EventArgs s)

{

Response.Write("<HI>Page\_Init()메서드가 호출됨<HI><P>");

//this.Load += new System.EventHandler(this.Page\_Load);

}

void Page\_Load(object sender, EventArgs s)

{

Response.Write("<HI>Page\_Load()메서드가 호출됨<HI><P>");

// this.Unload += new System.EventHandler(this.Page\_Unload);

}

void Page\_Unload(object sender, EventArgs s)

{

}

}

[ 반드시 알아야 할 웹 폼 이벤트 ]

Init **Load(가장 많이 사용) PreRender**

**웹 폼 이벤트 실행 순서**

- Init이벤트 ==> LoadViewState이벤트 ==> Load 이벤트

==> 서버 컨트롤 이벤트(ex:Button 컨트롤의 Click이벤트)

==> PreRender이벤트

==> SaveViewState 이벤트

==> Unload 이벤트

\*) Init이벤트 : LoadViewState 이벤트 이전에 발생

따라서 서버 컨트롤 개체를 사용할 수 없다.

\*) LoadViewState : 서버 컨트롤의 정보를 가지고 있는 뷰

- 이 이벤트 후부터 서버 컨트롤 개체 사용 가능

\*) Load : 초기화

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

protected void Page\_PreRender(object sender, EventArgs e) {

Response.Write("PreRender 이벤트 발생<br>");

}

protected void Page\_Init(object sender, EventArgs e) {

Response.Write("Init 이벤트 발생<br>");

}

protected void Page\_Load(object sender, EventArgs e) {

Response.Write("Load 이벤트 발생<br>");

}

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >

<head runat="server">

<title>제목 없음</title>

</head>

<body>

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

<div>

</div>

</form>

</body>

</html>

**[ Page 클래스의 IsPostBack 프로퍼티 ]**

- 클라이언트들은 페이지를 요청하고 적당한 입력을 하고 난 뒤에 입력된 내용을 전송하기 위해서 버튼

을 클릭함

이렇게 되면 웹 서버는 요청에 대한 응답을 보내기 위해서 다시 해당 페이지를 전송함

(이렇게 자신을 타겟으로 하여 현재 페이지를 다시 서버로 전송하는 프로세스를 다시게시-Postback라함)

여기에서 나타나는 문제점은 다음과 같은 코드에서 볼수 있다.

<%@ Page language="C#" %>

<script runat="server">

int count = 0;

void Page\_load(object Sender, EventArgs e)

{

labelTest.Text = "페이지를 처음 요청했습니다.";

labelCount.Text = "0";

}

void buttonTest\_Click(object Sender, EventArgs e)

{

labelTest.Text = "페이지의 재전송을 요청했습니다.";

count = Convert.ToInt32(labelCount.Text);

count++;

labelCount.Text = count.ToString();

}

</script>

<html>

<body>

<form id = "Form1" method="post" runat="server">

<asp:Label id="labelTest"

runat="server"/>

<P>

<asp:Label id="labelCount"

runat="server"/>

<P>

<asp:button id="buttonTest"

Text="클릭 하면 수가 증가됩니다."

OnClick="buttonTest\_Click"

runat="server"/>

</form>

</body>

</html>

클라이언트 버튼 클릭 ==> 웹 서버는 페이지 재전송 계속해서 Page\_load 발생

// 해결 방안?

void Page\_load(object Sender, EventArgs e)

{

if (!IsPostBack)

{

labelTest.Text = "페이지를 처음 요청했습니다.";

labelCount.Text = "0";

}

}

// IsPostBack : 해당 페이지가 처음 요청될때에는 false 재전송 요구하면 true로 설정됨

// 재 초기화를 막을 수 있다.

//예제2)다시 게시 : 현재 페이지를 다시 서버로 전송하는 프로세스

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

if (Page.IsPostBack)

{

Response.Write("다시 게시입니다.");

}

else

{

Response.Write("다시 게시가 아닙니다.");

}

}

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >

<head id="Head1" runat="server">

<title>제목 없음</title>

</head>

<body>

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

<div>

<asp:Button ID="Button1" runat="server" Text="Button" />

</div>

</form>

</body>

</html>

//예제3) 다시게시를 통해 발생되는 문제점

- Init이벤트 ==> LoadViewState이벤트 ==> Load 이벤트

==> 서버 컨트롤 이벤트(ex:Button 컨트롤의 Click이벤트)

==> PreRender이벤트

==> SaveViewState 이벤트

==> Unload 이벤트

\*\*) 페이지가 처음 로드되거나 버튼 클릭시 다시 게시를 일으킬 때 Load 이벤트는 항상 발생

서버 컨트롤 이벤트(버튼 컨트롤)는 다시 게시될때 발생되며 Load이벤트보다 나중에 발생된다.

사용자가 문자열을 수정하여도 Load이벤트 처리기가 계속적으로 발생되어 계속 초기화 하게 된다.

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

TextBox1.Text = "초기화된 문자열";

}

protected void Button1\_Click(object sender, EventArgs e)

{

Response.Write(TextBox1.Text);

}

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >

<head id="Head1" runat="server">

<title>제목 없음</title>

</head>

<body>

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

<div>

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<asp:Button ID="Button1" runat="server" Text="확인" OnClick="Button1\_Click" />

</div>

</form>

</body>

</html>

// 해결 방법

protected void Page\_Load(object sender, EventArgs e)

{

if( !Page.IsPostBack)

TextBox1.Text = "초기화된 문자열";

}

//예제4) 페이지간 게시

* Ex01.aspx 파일
* 페이지간 게시 : 웹폼 🡺 다른 웹폼으로 서버 컨트롤의 정보 전달

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

protected void Button1\_Click(object sender, EventArgs e)

{

Response.Write("다시 게시: " + TextBox1.Text);

}

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >

<head id="Head1" runat="server">

<title>제목 없음</title>

</head>

<body>

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

<div>

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<asp:Button ID="Button1" runat="server" Text="다시 게시" OnClick="Button1\_Click" />

<asp:Button ID="Button2" runat="server" Text="페이지 간 게시" PostBackUrl="Default3.aspx" />

</div>

</form>

</body>

</html>

// 또 다른 웹 폼 .cs 파일

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

if (PreviousPage != null)

{

TextBox t;

t = (TextBox)PreviousPage.FindControl("TextBox1");

Response.Write("페이지 간 게시: " + t.Text);

}

else

{

Response.Write("페이지 간 게시가 아닙니다.");

}

}

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >

<head id="Head1" runat="server">

<title>제목 없음</title>

</head>

<body>

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

<div>

</div>

</form>

</body>

</html>

**[ HTML 컨트롤 ]**

- 기존의 HTML 태그와 거의 대부분이 일대일 매핑되어 있다.

- 이 컨트롤 사용 이유

HTML 태그 속성을 런타임 시에 쉽게 조작할 수 있다.

객체 지향적인 디자인을 가능케 해준다.

- 관계

HTML 컨트롤 HTML 태그

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

HtmlAnchor <a>

HtmlButton <button>

HtmlSelect <select>

HtmlTextArea <textarea>

HtmlInputButton <input type="button">

HtmlInputCheckBox <input type="check">

HtmlInputText <input type="text>, <input type="password">

HtmlInputHidden <input type="hidden">

HtmlInputImage <input type="image">

HtmlInputFile <input type="file">

HtmlForm <form>

HtmlImage <img>

HtmlTable <table>

HtmlTableRow <tr>

HtmlTableCell <td>

HtmlGenericControl <span>, <div> , <body> , <font> 등

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

- HTML 컨트롤은 기존 HTML 태그와 달리 서버에서 실행될 수 있는 서버 컨트롤이다.

- 브라우저의 종류와 버전을 체크하지 않는다.

따라서 동일한 HTML 결과만 생성한다.

// ex1) html 컨트롤 이용 예제

<%@ Page language="C#" %>

<script runat="server">

void htmlInputButtonTest\_ServerClick(object sender, EventArgs s)

{

htmlInputButtonTest.Value = "클릭하셨습니다";

}

</script>

<html>

<body>

<form id = "Form1" method="fost" runat="server">

<input type="button"

id="htmlInputButtonTest"

value ="클릭하세요"

OnserverClick = "htmlInputButtonTest\_ServerClick"

runat="server" />

</form>

</body>

</html>

- 각 HTML 컨트롤 클래스는 HtmlControl 클래스에서 파생된다

- 이 클래스는 HTML 컨트롤의 일반적인 메서드나 프로퍼티를 제공한다.

- 많은 HTML 컨트롤들은 이벤트를 노출할 수 있다.

**[ 웹 컨트롤 ]**

- System.Web.UI.WebControls 네임스페이스에 속해 있다.

- HTML태그를 확장한 것 같은 형식임

- 겉으로 보기에는 HTML 서버 컨트롤과 유사하지만

웹폼을 이용하면 완전한 컨트롤 프로그래밍이 가능하고 클라이언트의 종류와 버전을 체크하여 알맞는

HTML결과를 생성한다.

- ex)

클라이언트가 휴대폰이라면?

웹 폼 컨트롤에 의해서 생성된 코드는 WML과 같은 코드로 변환됨

자동으로 이루어짐

따라서 클라이언트마다 따로 개발할 필요가 없이

하나의 웹 어플리케이션만 개발하면 모든 클라이언트를 수용할 수 있다.

- ASP.NET에서 가장 핵심 요소임

- 특징

1) 어플리케이션 로직과 HTML 인터페이스를 완전히 분리

2) 서버측에서 수행되고, 클라이언트가 사용하는 브라우저의 종류와 버전을 알아내서, 적절한 html코드

를 생성해서 전달함

3) 데이터베이스를 다루는 웹 컨트롤들을 이용해서 코드 양을 줄임

4) 완전한 이벤트기반 프로그래밍 가능

5) 개발자가 직접 웹 폼 및 웹 컨트롤들을 작성해서 재사용가능

- 모든 웹 폼 컨트롤들은 Webcontrol 클래스를 상속함

- 웹 폼 컨트롤이 가지고 있는 공통된 프로퍼티

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

프로퍼티 네임 타입 의미

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

BackColor Color 웹 컨트로의 배경색을 나타냄

Enabled bool 웹 컨트롤의 활성화 여부를 나타냄

Font FontInfo 웹 컨트롤의 폰트 정보를 리턴

ForeColor Color 웹 컨트롤의 전경색을 나타냄

Height Uint 웹 컨트롤의 높이를 나타냄

ID string 웹 컨트롤을 프로그래밍 할때 명시자

Page Page 웹 컨트롤을 포함하는 페이지를 리턴

Style CssStyleCollection 웹 컨트롤의 스타일 리턴

ToolTip string 웹 컨트롤의 툴팁 나타냄

Visible bool 웹 컨트롤을 페이지에 보이게 할지 여부

Width Uint 웹 컨트롤의 너비를 나타냄

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

- 웹 컨트롤들은 일반적인 HTML태그와 동일한 기능을 하는것도 있지만

웹 폼 컨트롤은 HTML 태그에 존재하지 않는 새롭고 강력한 컨트롤들을 가지고 있다.

웹 컨트롤의 종류

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

웹 컨트롤 사용되는 이벤트 의미

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

Label 없다 텍스트를 나타냄

TextBox TextChanged User의 텍스트 입력

Button Click, Command 버튼 표시

LinkButton Click, Command 링크 표시

ImageButton Click 이미지를 버튼처럼 사용

HyperLink 없다. 하이퍼링크 표시

DropDownList SelectedIndexChanged 콤보박스

ListBox SelectedIndexChanged 리스트박스

Image 없다 이미지 표시

AdRotator AdCreated 베너 광고

CheckBox CheckChanged 여러개 선택

CheckBoxList SelectedIndexChanged 여러개의 체크박스를 관리

RadioButton CheckChanged 라디오버튼을 나타냄

Calender SelectionChanged,

VisibleMonthChanged,

DayReder 달력

Table 없다 표 혹은 테이블

TableCell 없다 테이블의 셀

TableRow 없다 테이블의 열

Panel 없다. 또 다른 컨트롤을 포함

Repeater ItemCommand,

ItemCreated,

ItemDataBound 테이블을 바운드하는 가장

기본적인 컨트롤

DataList CancelCommand,

EditCommand,

DeleteCommand,

ItemCommand,

SelectedInexChanged,

UpdateCommand,

ItemCreated,

ItemDataBound 데이터를 리스트 형식으로

나타내고 관리

DataGrid CancelCommand,

EditCommand,

DeleteCommand,

ItemCommand,

SelectedIndexChanged,

PageIndexChanged,

SortCommand,

UpdateCommand,

ItemCreated, 데이터들을 그리드형식으로

ItemDataBound 나타태고 관리

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

- 웹 컨트롤 사용

1) 추가

HTML과 마찬가지로 태그-기반의 구문을 사용

웹 컨트롤은 asp: 을 항상 앞에 붙임

Button 웹 컨트롤 선언

<asp:button id="Write" Text="쓰기" OnClick="Write Click" runat="server"/>

or

<asp:button id="Write" Text="쓰기" OnClick="Write Click" runat="server">

</asp:button>

ex1) 웹 컨트롤

<%@ Page language="C#" %>

<%@ Import Namespace="System.Drawing" %>

<script runat="server">

void buttonTest\_Click(object Sender, EventArgs e)

{

if (labelTest.ForeColor == Color.White ||

labelTest.BackColor == Color.Black)

{

labelTest.ForeColor = Color.Black;

labelTest.BackColor = Color.White;

}

else

{

labelTest.ForeColor = Color.White;

labelTest.BackColor = Color.Black;

}

}

</script>

<html>

<body>

<form id = "Form1" method="post" runat="server">

<asp:button id="labelTest"

Text="Label 웹 컨트롤이다."

runat="server"/>

<asp:button id="buttonTest"

Text="클릭 하면 색이 반전된다."

OnClick="buttonTest\_Click"

runat="server"/>

</form>

</body>

</html>

//- color 구조체를 사용하기 위해서 import 처리

**[ ASP.NET 페이지의 요소 ]**

- 앞에서 살펴본 것과 같이 다양한 요소로 이루어져 있슴

|  |  |
| --- | --- |
| 요소 | **의미** |
| 정적 HTML태그 | 표준 HTML요소들을 의미, 클라이언트 브라우저에서 랜더링 됨 |
| HTML주석 | <!---->  클라이언트에게 그대로 보내지지만 브라우저가 랜더링 하지 않는다 |
| 지시어 | @Page와 같은 지시어  해당 페이지가 처리되는방법이나 정보를 ASP.NET 런타임에게 제공  세션상태, 이벤트 자동 연결 여부, 출력 캐싱, 네임스페이스 설정 등과 같은  특성 조작 가능 |
| 서버-사이드코드 | <script runat="server"> or <%, %> 사이 코드  서버에서 랜더링 된다 |
| 이벤트 핸들러 | <script> 코드 내에 선언됨  Page\_Load() 메서드 등을 말함 |
| 클라이언트-  사이드코드 | <script> </script> 사이의 코드를 의미  자바스크립트와 같이 클라이언트 브라우저에서 실행되는 코드를 의미함 |
| 서버-사이드 주석 | <%-- --%>  HTML주석과 달리 클라이언트에게 전달되지 않음 |
| 사용자 컨트롤 | .ascx 확장자를 가진 파일에 정의된 커스텀 컨트롤  개발자가 만든 컨트롤을 재사용하기 위해 만든 컨트롤 |
| ASP.NET 서버컨트롤 | 기존의 GUI 프로그래밍 모델과 유사하게 ASP.NET에서 사용할 수 있도록 제  공되는 내장 컨트롤  ASP.NET에서는 HTML컨트롤과 웹 컨트롤로 구분됨 |
| HTML 컨트롤 | 표준 HTML 요소들과 일대일 매핑되는 컨트롤 |
| 웹 컨트롤 | 윈도우 폼 컨트롤과 유사하게 프로그래밍 할 수 있는 컨트롤  이것은 이벤트 방식의 프로그래밍을 가능케 함 |

- ASP.NET 웹 폼에 사용되는 많은 요소들이 존재하고 기능을 정리해 보자

**[ 숨겨진 \_VIEWSTATE ]**

- ex) 회원가입

- 회원 가입을 위한 정보 입력

- 전송 버튼 클릭

- 만약 에러가 발생된다면??

- User는 전페이지로 이동함

- 내용이 다 삭제되어 있슴

=> ASP.NET에서는 HIDDEN 타입의 컨트롤을 추가해서 페이지의 상태 변화를 저장하고,

다음에 다시 페이지를 요청해도 계속 상태가 유지되도록 한다.

클라이언트가 입력한 값이 서버에 저장되는 것이 아니라 전달받은 HTML 코드에 저장되어 있으

므로 효율적이다.

**[ VisualStudio.net을 이용한 ASP.NET 프로그래밍 ]**

- 장 점

1) 여러 개의 파일들을 하나의 프로젝트로 관리

2) 웹 폼 서버 컨트롤을 매우 쉽게 다룰 수 있다.

3) 데이터베이스를 이용한 어플리케이션을 쉽게 다룰 수 있다.

4) 웹 어플리케이션들에 대한 배포가 용이하다.

5) 다양한 프로그래밍 언어를 하나의 통합 개발 환경에서 다룰 수 있다.

\*) 테스트

[도구 상자] – [Web Froms] 에서 Label 컨트롤을 마우스를 이용해 디자인 화면 적당한 위치에 놓음

WebForm1.aspx

* HTML 코드 확인 : 페이지 하단의 HTML 버튼 클릭
* 코드 비하인드 방식 지원

유용한 기능

1. 디버깅...

프로그래밍 오류 문법 오류 논리 오류

// 예제1) 논리 오류 예제 : 30을 10으로 나눈다면??

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">

protected void Button1\_Click(object sender, EventArgs e)

{

double d1 = double.Parse(TextBox1.Text);

double d2 = double.Parse(TextBox2.Text);

double result = Divide(d1, d2);

Response.Write(result);

}

double Divide(double param1, double param2)

{

return param2 / param1;

}

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >

<head id="Head1" runat="server">

<title>Untitled Page</title>

</head>

<body>

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

<div>

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox> /

<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

<asp:Button ID="Button1" runat="server" Text="계산" OnClick="Button1\_Click" />

</div>

</form>

</body>

</html>