2nd Generation Server-Side Scripting Environments

Server Side Scripting Pros & Cons

- First generation server-side scripting environments such as PHP, Classic ASP, and traditional ColdFusion enjoyed a great success because they were relatively easy for both new and experiences programmers to pick up.
- First generation server-side scripting environments do have challenges including:
  - Performance issues
  - Heavy intermixture of script and markup
  - Lack of facilities for large scale system development
    - Debugging
    - Modern coding features like OOP, exception handling, strong typing

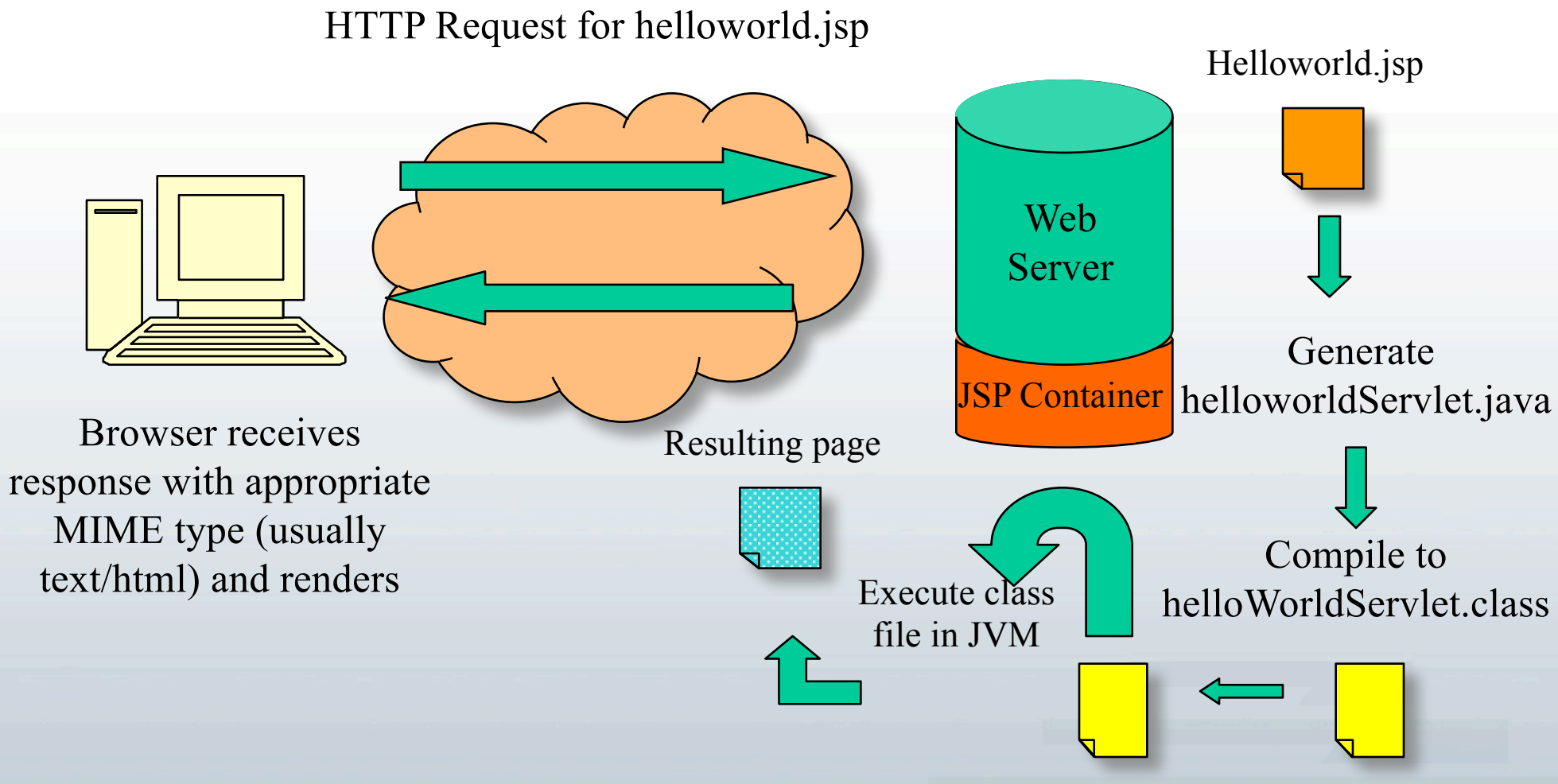
Second Generation Server-side Scripting

- The Java Web platform provides JSP while the .NET platform provides ASP.NET as potential 2nd generation server-side scripting environments
- Both address speed through pre-compilation to intermediate code
  - Performance hit on first request, but significant savings later
- Both provide more support for large scale system development (generally through improved object access, language features, etc.)
- Both try to get developers not to intermix code and markup
  - Easier said than done

Intro to Java Server Pages

- Java Server Pages or JSP (java.sun.com/products/jsp) for short provide a scripting approach for building Java based web pages
  - Scripting and element based content
  - Compiled pages for better execution
  - Can use servlets
  - Leverages the Java environment
    - Rich library, type safety, OOP focused development
- Need a JSP enabled Web server
  - Try Apache Tomcat (http://jakarta.apache.org/tomcat)
  - Plenty of other JSP/Servlet hosts (http://www.servlets.com/engines)

Visual Overview of How JSP Works



How JSP Works

- As long as original JSP source is unchanged, subsequent requests go right to the Servlet Class file – thus the performance boost

- In some ways you might say that JSP is a simpler way for someone to write a Servlet

JSP Elements

- There are three types of elements in JSP
  1. Directive `<%@ %>`
  2. Action `<jsp: >`
  3. Scripting `<% %>` , `<%= %>`, and `<%! %>`

- Expression Language (EL) is also found in JSP pages and is a simple JavaScript like language for simple coding

- JSP comments
  - `<%-- --%>`

JSP Directive Elements

- Directive elements specify information about the page and its processing.

- There are three typical directives
  1. <%@ page … %> - defines page attributes such as session tracking, error page, and buffering
  2. <%@ include .. %> - includes a file
  3. <%@ taglib … %> - declares a custom tag library to be used within the page

JSP Standard Action Elements

- Action elements are called upon at page request time to do general things as shown by the examples here:
  1. <jsp:useBean> - makes a JavaBeans component available in page
  2. <jsp:getProperty> - gets a property value from a Java Bean
  3. <jsp:setProperty> - sets a property value for a Java Bean
  4. <jsp:forward> - forwards the request to a a servlet or JSP page specified
  5. <jsp:include> - includes a response from another JSP or servlet in the page
  6. <jsp:param> - adds a parameter value to a request handed off to another JSP or servlet using <jsp:include> or <jsp:forward>
  7. <jsp:plugin> - generates HTML to execute a Java applet

JSTL and Custom Action Elements

- The standard actions perform very specific Web application oriented tasks so we may utilize tags from the JSP Standard Tag Library (JSTL) - http://java.sun.com/products/jsp/jstl/ or we can make our own custom tags.

- JSTL contains five categories :
  1. Core – conditional processing, looping, data import, etc. (prefix c)
  2. XML (prefix x)
  3. Internalization (I18N) and formatting (prefix fmt)
  4. Relational Database Access (prefix sql)
  5. Functions (prefix fn)

Scripting Elements

- Scripting elements allow you to add small pieces of Java code in a page
  - <% … %> used to embed some code
    -
```
<%
        out.println("<h1>Hello world</h1>");
%>
```
  - <%= %> when you want to just output something or set a value often used in attributes
    - `<%= new java.util.Date() %>`
  - <%! … %> used to set values
    - `<%! String name = "Thomas"; %>`

Hello World in JSP

- Numerous ways to write this intro example:
  - Embedded Java print statement
    - <%
      out.println("<h1>Hello world</h1>");
      %>
  - Embedded literal <%= %>
    - <%= "<h1>Hello world</h1>" %>
  - Using JSTL Core tag <c:out>
    - <c:out value = "<h1>Hello world</h1>"; %>

Hello World in JSP - 5 ways

```
<%@ page language="java" contentType="text/html" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html><head><title>Hello JSP World</title></head>
<body>
<%
  out.println("<h1>Hello world from JSP Take 1</h1>");
%>
<%= "<h1>Hello world from JSP Take 2</h1>" %>

<h1>
  <c:out value="Hello world from JSP Take 3" />
</h1>

<c:out value="<h1>Hello world from JSP Take 4</h1>"
    escapeXml="false" />
<c:out value="<h1>Hello world from JSP Take 5 </h1>" />

<hr />
All these greetings took place at: <%= new java.util.Date() %>
</body></html>
```

Running Hello World

- If we type this in and try it the first time you will notice a significant delay as a servlet code is generated then a class file compiled and executed via the JVM
  - Further requests are fast though
- Given that JSP makes a servlet let's take a look at a similar helloworld servlet

Now as a Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class helloworld extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse
     res)
                throws ServletException, IOException {

       res.setContentType("text/html");
       PrintWriter out = res.getWriter();
       out.println("<html>");
       out.println("<head><title>Hello World</title></head>");
       out.println("<body");
       out.println("<h1>Hello World directly from a
     Servlet</h1>");
       out.println("</body></html>");
   }
}
```

Preparing the Servlet

- Compile the helloworld.java file to helloworld.class
- Make sure you have a special directory structure for your servlet –
  - app-path (arbitrary directory often under webapps directory)
    - /WEB-INF
      - /classes (put your code in here)
      - /lib (put any JAR files in here)
      - web.xml (describes the servlets and related issues)
  - An example web.xml file is shown on the next slide

Preparing the Servlet Contd. – web.xml example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app   PUBLIC "-//Sun Microsystems, Inc.//DTD Web
      Application 2.3//EN"
   "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
   <display-name>Hello World Servlet</display-name>
   <description>The famous Hello World as a servlet</description>
   <servlet>
     <servlet-name>helloworld</servlet-name>
     <description>Testing</description>
     <servlet-class>helloworld</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>helloworld</servlet-name>
      <url-pattern>/helloworld</url-pattern>
   </servlet-mapping>
</web-app>
```

Preparing the Servlet Contd. – web.xml example

- Deploying the servlet can often be more difficult than some of the basic code
- Because of this some will use Ant to automate the build process particularly if you want to write code in one directory and deploy in another
- A WAR file (Web application archive) helps address deployment issues
  - WAR file is just a rename of a JAR file, save that the internal structure has to map to the WEB-INF style previously mentioned
- Also be aware that you may have to restart your Java server environment or access its management system to reload any servlets you may modify
- *Notice here a slight difference – trigger the app via the URL over the file extension*

Servlets and JSP – The Same Old Story

- Notice that in the Servlet example we output the MIME type and the various HTML statements (shades of CGI) while in the JSP example that is taken care of by the scripting environment (just like PHP, etc.)
- Further notice having to know many server and HTTP details to code a servlet versus JSP which hides much of this
- It would appear server-side programming wise that the more things appear to be different the more it turns out they are the same
  - "Code prints HTML or HTML that runs code"
  - "Hard to implement often equals run faster"

JSP Expression Language

- JSTL 1.0 introduces the simple Expression Language (EL) for setting attribute values and doing basic runtime calculations
  - Somewhat ECMAScript like
  - Delimited by ${ }
  - Lacks control structures – use action elements from JSTL for that

- Simple example
  - <c:out value="${5 * 5}" />
  - Or just
    ${5*5}
    right in the page

JSP Expression Language Overview

- Operators
  - `.` Access a property of a Bean or other object
  - `[ ]` - array access
  - `( )` - grouping
  - `? :` - standard ternary conditional operator
  - `+` - Addition
  - `–` - Subtraction
  - `*` - multiplication
  - `/` *or* `div` - division
  - `%` *or* `mod` - modulus
  - `==` *or* `eq` - equality
  - `!=` *or* `ne` - not equal

JSP Expression Language Overview

- More operators
  - `<` *or* `lt` - less than
  - `>` *or* `gt` - greater than
  - `<=` *or* `lte` - less than equal to
  - `>=` *or* `gte` - greater than or equal to
  - `&&` *or* `and` - logical and
  - `||` *or* `or` - local not
  - `!` *or* `not` - not
  - `func(`*`arg`*`)` - function call to Func

JSP Expression Language Overview

- Numerous implicit variables available which should seem to make sense including:

```
pageScope
requestScope
sessionScope
applicationScope
param
paramValues
header
headerValues
cookie
```

- Any guesses what

```
<c:out value="${param.userName}" />
```

would be used to output?

JSTL Core Library Actions

- <c:catch> - basic exception handling
- <c:choose>
  <c:when action>
  <c:otherwise>
  </c:choose> - a switch/case statement
- <c:forEach> - an iterator
- <c:forTokens> - another iterator
- <c:if> - basic selection
- <c:import> - file inclusion
- <c:out> - basic output
- <c:rediret> - page redirection
- <c:set> - sets a variable (can define a scope)
- <c:remove> - removes a variable
- <c:url> - encodes a URL
- <c:param> - used to pass "query parameters" to <c:url>, <c:include>, or <c:redirect>

Example: Basic Form Handling

```
<html>
<head>
<title>Simple Form</title>
</head>
<body>
<h1>Enter Your Data to Receive your SPAM</h1>
<hr>
<form action="collect.jsp" method="post">
<label>Name:<input type="text" name="username"></label><br>
<label>Email:<input type="text" name="email"></label><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Example: Basic Form Handling JSP Handler

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="c"
   uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
<title>SPAM Confirm</title>
</head>
<body>

<h1>Thanks for your entry ${param.username}!</h1>
<p>Spam will be sent to ${param.email} ASAP!</p>

</body>
</html>
```

Example: Simple Header Print

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="c"
   uri="http://java.sun.com/jsp/jstl/core" %>
<html><head><title>Environment and
   Request</title></head>
<body><h1>Headers</h1>
<c:forEach items="${headerValues}" var="h">
   <b><c:out value="${h.key}" /></b>;
   <c:forEach items="${h.value}" var="value">
      <c:out value="${value}" />
   </c:forEach>
<br>
</c:forEach>
</body></html>
```

Tags – Better than they look?

- As clunky as that last example looked what makes it potentially a good idea?

- Question: Could you validate a template with code all over the place easily?
  - Consider if that code splits across tags

- Consider the idea of separation of duties – developer, designer, content, etc.

ASP.NET Overview

What the heck is .NET?

- Good question
- The main "components" of .NET include:
  - A runtime system more specifically a Common Language Runtime, or CLR
  - A large set of classes that serve as an API for program's run through the CLR
    - These are referred to as the .NET Framework classes or the .NET SDK
  - New programming languages such as C#
- Like many Microsoft technology initiatives (DNA, ActiveX, etc.) the grandiose everything under one name actually makes things quite confusing

What the heck is .NET? Contd.

- Like Java you write code which is converted into an intermediate form.
- In the case of .NET the intermediate language is called MSIL
  - high-level, architecture independent assembly language
- So a .NET compiler would take VB.NET, C#, etc. source code and convert to MSIL
- The CLR then takes the MSIL and converts to platform specific code (Just-in-Time compilation)
  - JITed copies run later on without the transformation overhead

.NET Programming Languages

- To create .NET programs, you need to use a .NET programming language (one that compiles to MSIL and uses the .NET Framework classes). Such languages include:
  - JScript.NET
  - VB.NET
  - C#
  - Others
- ASP.NET uses one of these languages
  - *Remember ASP and ASP.net is a framework not a language though it is commonly associated that way*

C#

- C# (pronounced C-Sharp) is the popular.NET language
- Structure and syntax closely resembles that of Java and C/C++
  - Goal with C# to take the good features of Java and C++ and dump the bad ones
    - No pointers
    - Very Object-oriented
    - Strong typing
    - It keeps evolving though…concerns?

.NET Language Choice

- In theory language choice in .NET is more based upon familiarity than anything
  - All languages should have access to the same class framework
  - In practice C# is quite favored
  - All languages share similar data types
    - Intermixing languages should even be possible
  - Because of MSIL architecture all should have pretty much same performance
    - Maybe not? This might be a function of the compiler

ASP.NET

- Like classic ASP, ASP.NET (www.asp.net) sites are language independent
- Unlike classic ASP, ASP.NET pages are compiled .NET programs created in a .NET language
- 2nd Generation language goal: separation of code and content
  - ASP.NET solves this with
    - "Code-behind pages"
    - Server-controls
- ASP.NET also supports Web services

Comparing ASP.NET and ASP

Classic ASP	ASP.NET
ASP pages intermixed with a particular user defined form of markup	ASP.NET uses Web controls that return potentially device focused markup
Could use ADO to access database information.	Can access databases via ADO.NET.
Create ASP pages quickly and easily using VBScript/Jscript	Create ASP.NET quickly using VB.NET/C# in conjunction with a variety of controls.

Contrasting ASP.NET and ASP

Classic ASP	ASP.NET
Uses scripting languages (VBScript / JScript/PerlScript)	Uses compiled, .NET-compatible language (VB.NET / C# / JScript.NET, etc.)
Web pages contain .asp extension	Web pages contain .aspx extension.
markup and server-side script intermixed.	Server-side script placed in distinct <script> blocks or even separate source files.

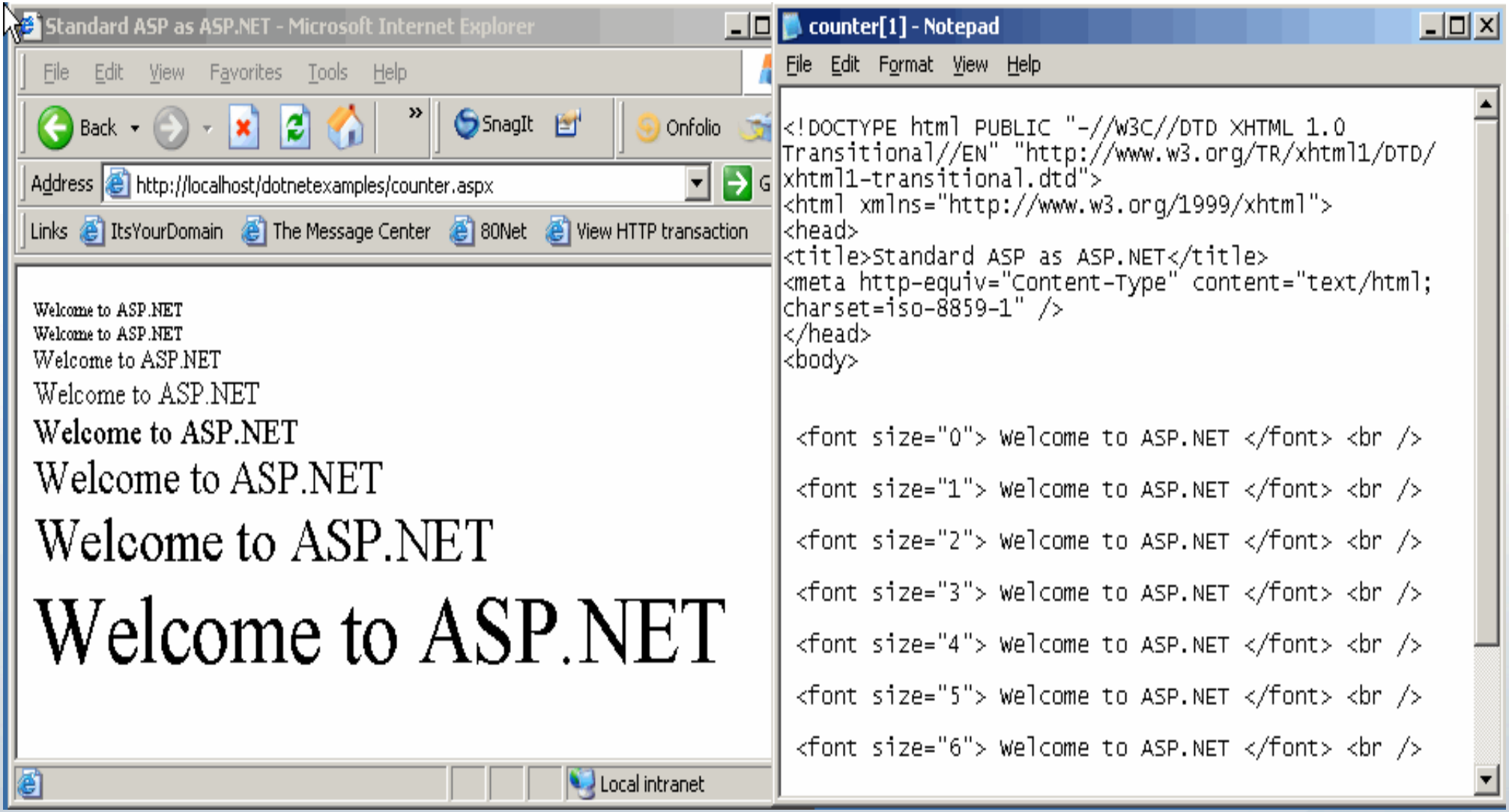
Structure of ASP.NET Pages

- End with .aspx extension
- Three distinct sections
  - Page directives – how to process page, language, other runtime considerations
  - Code section - <script> block with code to run
  - Actual page template – the template that gets filled in
- However you can also use the standard ASP style of interpreted script

Simple ASP style ASP.NET page

```
<%@ Page Language="VB" ContentType="text/html"
    ResponseEncoding="iso-8859-1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Standard ASP as ASP.NET</title>
</head>
<body>
<% Dim I As Integer
    For I = 0 to 7 %>
 <font size="<%=I%>"> Welcome to ASP.NET </font> <br />
<% Next %>
</body>
</html>
```

Simple Example Rendering



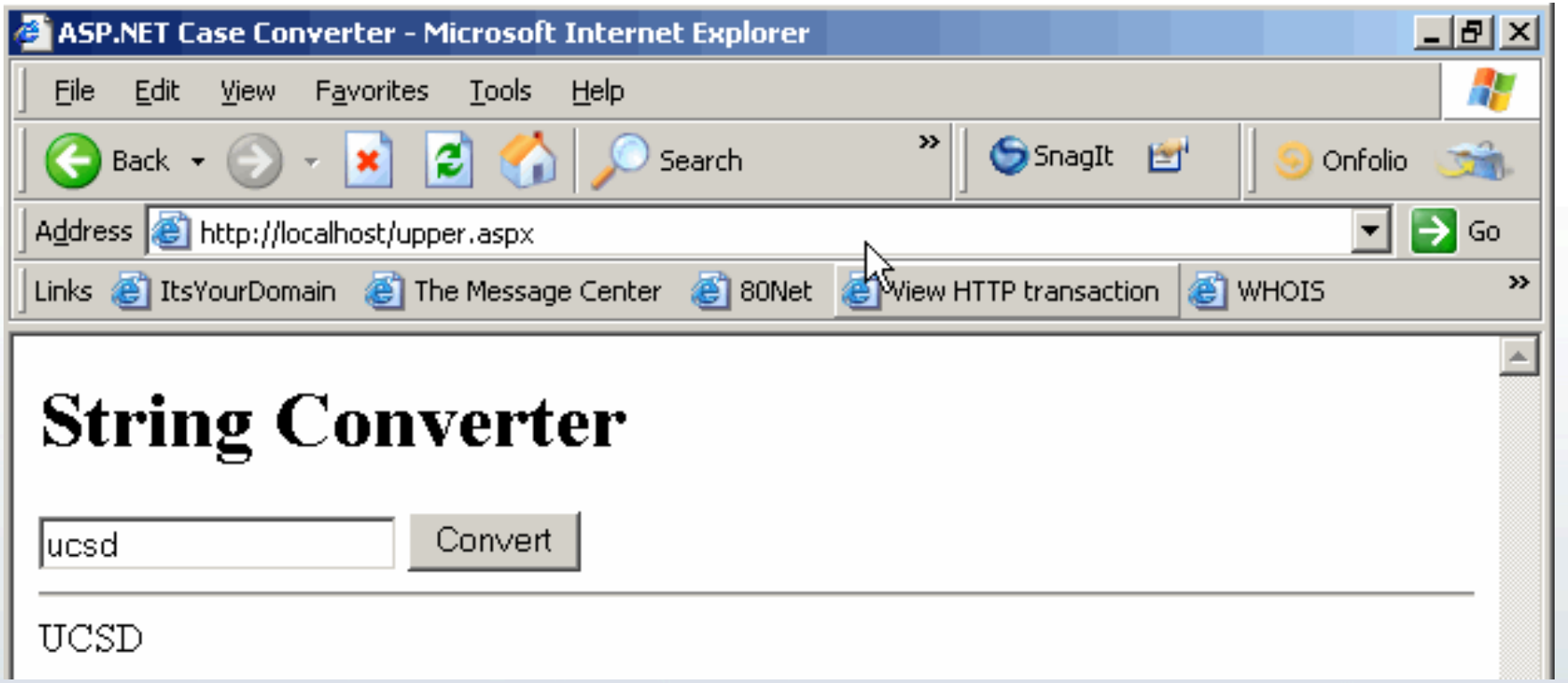
Comments on the First Example

- Of course this inline code provides little of the benefit of ASP.NET
  - No compilation, instead embedded into page logic and executed only at render time
    - There are of course good reasons for this
  - It doesn't separate the code and content like we wanted to
- The next example gets us a little farther separating out the code somewhat and showing the WebForms model a little bit and a few simple server controls

A Little Better ASP.NET Example

```
<%@ Page Language="C#" ContentType="text/html" ResponseEncoding="iso-8859-1" %>
<script runat="server">
private void MakeUpper(object sender, EventArgs e) {
  string buf = TheString.Text;
  TheResult.Text = buf.ToUpper();
}
</script>
<!DOCTYPE html>
<html>
<head><title>ASP.NET Case Converter</title></head>
<body>
<h1>String Converter</h1>
<form runat="server" id="TheForm">
     <asp:textbox runat="server" id="TheString" />
     <asp:button runat="server" text="Convert" onclick="MakeUpper" />
      <hr />
    <asp:label runat="server" id="TheResult" />
</form>
</body></html>
```

Simple Example Rendering



Compiled Page Result

```
<!DOCTYPE html>
<html>
<head><title>ASP.NET Case Converter</title></head>
<body>
<h1>String Converter</h1>
<form name="_ctl0" method="post" action="upper.aspx"
   id="_ctl0">
<input type="hidden" name="__VIEWSTATE"
   value="dDwtMTA0MDM0ODg4Nzs7Pu0WLg6iP1YIUWwr5YvKidrPcd1h"
   />
    <input name="TheString" type="text" value="ucsd" id="TheString" />
    <input type="submit" name="_ctl0" value="Convert" />
     <hr />
   <span id="TheResult">UCSD</span>
</form>
</body></html>
```

A few notes on the last example

- You'll note that ASP.NET server controls automatically maintain any client-entered values between round trips to the server.
  - This is done with the VIEWSTATE hidden field
  - Problem?
    - Size?
    - Security?
- Traditionally much of .NET is server-side
  - Why?
  - Post-back fun!
- Modern ASP.NET is Ajax driven mostly now

Simple Example with Code Behind

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
    AutoEventWireup="false" Inherits="upper2.WebForm1" %>
<!DOCTYPE HTML>
<html>
<head>
<title>WebForm1</title>
<meta name="CODE_LANGUAGE" Content="C#">
</head>
<body MS_POSITIONING="GridLayout">
<form runat="server" id="TheForm">
<asp:textbox runat="server" id="TheString" />
<asp:button runat="server" text="Convert" id="Button1" name="Button1"/>
<hr>
<asp:label runat="server" id="TheResult" />
</form>
</body>
</html>
```

The start of the Code Behind – just a few declarations

- The example was laid out in Visual Studio .NET and the code behind was partially generated by this program
  - Good thing! Lots of code

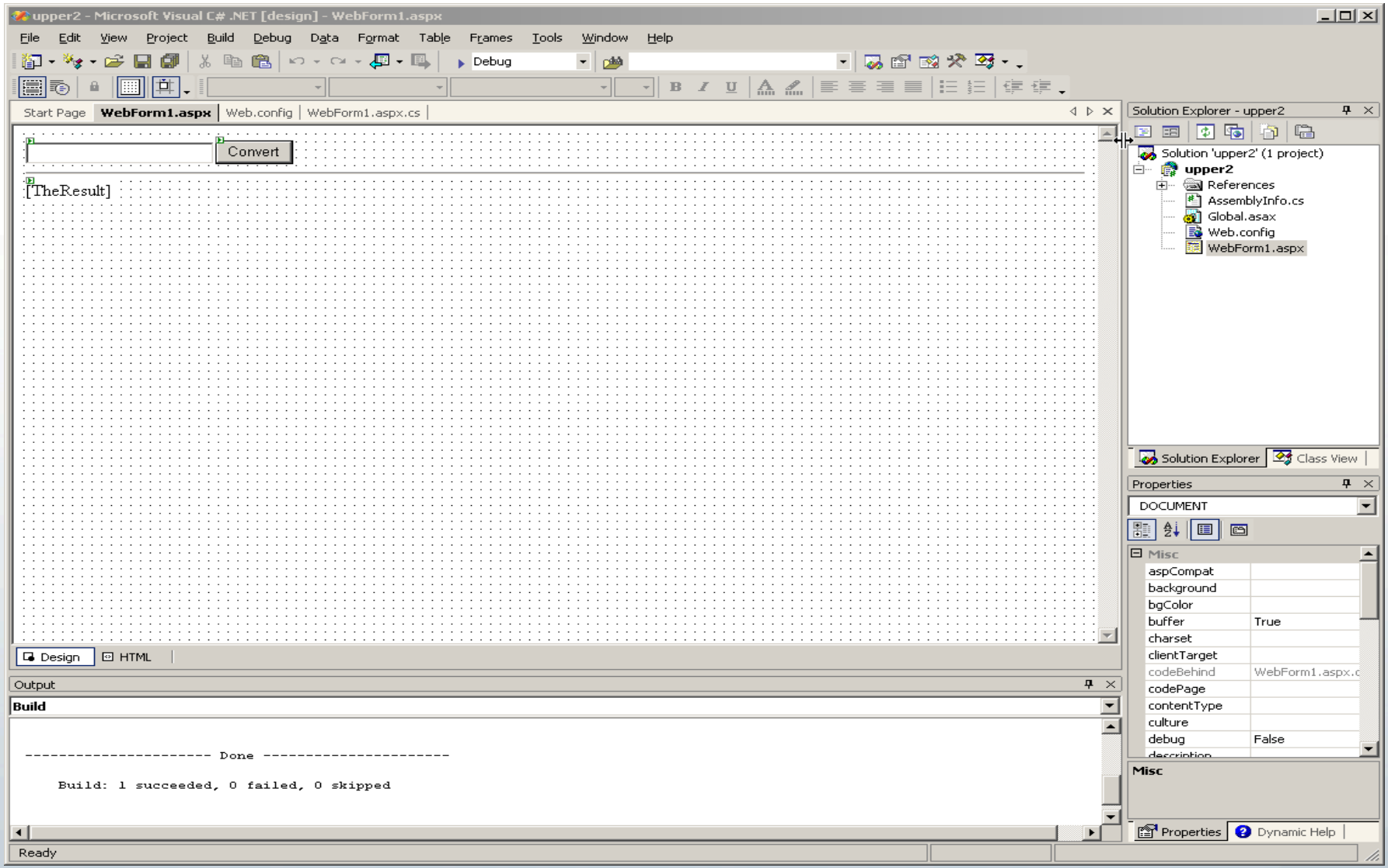
- The file WebForm1.aspx.cs starts first with just a few declarations

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
```

And the rest of the code

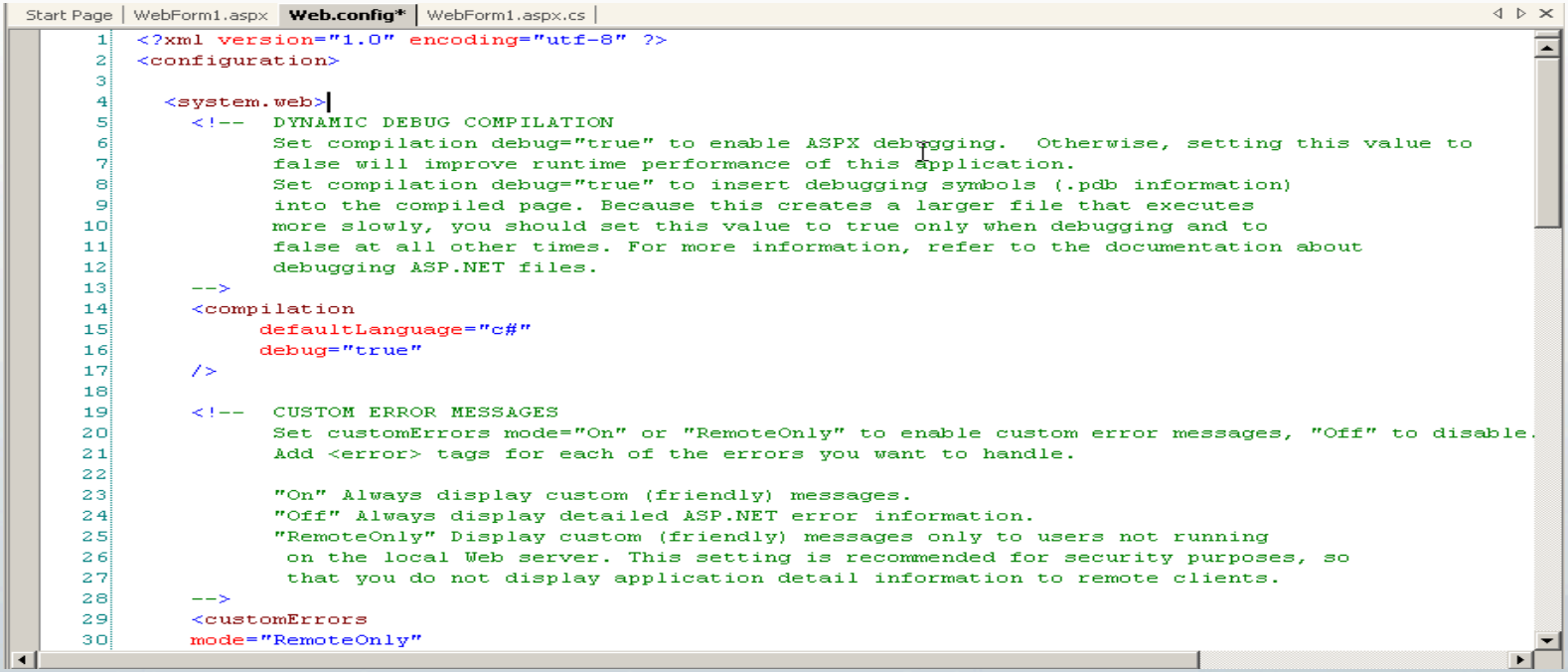
```
namespace upper2  {
      public class WebForm1 : System.Web.UI.Page {
        protected System.Web.UI.WebControls.TextBox TheString;
        protected System.Web.UI.WebControls.Button Button1;
        protected System.Web.UI.WebControls.Label TheResult;

        private void Page_Load(object sender, System.EventArgs e) {
                // Put user code to initialize the page here
        }
        override protected void OnInit(EventArgs e) {
          InitializeComponent();
          base.OnInit(e);
         }
       private void InitializeComponent() {
          this.Button1.Click += new System.EventHandler(this.Button1_Click);
          this.Load += new System.EventHandler(this.Page_Load);
        }
        private void Button1_Click(object sender, System.EventArgs e) {
            string buf = TheString.Text;
            TheResult.Text = buf.ToUpper();
        }
      }
}
```

An IDE Makes It Much Easier



Is it that different?

- Much of the architecture of ASP.NET is very similar to nature to JSP, Beans and Servlets
  - Custom tags, byte code, dlls, Web.config



Going Farther with ASP.NET

- Ajax really throws a monkey wrench into the equation
  - When you are changing the state of the page without updating the viewstate the new state is not reflected on final submission
  - This is a problem with doing an client-side coding at all in a server oriented framework like ASP.NET
    - Q: Is this by design? Is this necessarily a good thing?
    - There are ways around this both going the MS route and not
- Newer additions of ASP.net have embraced the Ajax pattern
  - Today are starting to look more and more like a framework to build JS->REST style systems

Second Generation Conclusions

- Address speed in similar manners – try to get more wired into the Web server like a module but pay some price for that
  - First request hit (then how about pre-cache?)
  - Complexity
- Address large system dev issues in a similar fashion as well
  - Encourage separation of code, template, and configuration
  - Provide large framework of components to utilize
  - Suffer from complexity (learnability) and lock-in
    - *Q: So what is my suggestion you wonder?*

Some Ponderings

- Why so many languages?
  - Hard to get good at one
- Many pieces = many programmers
- JavaScript is immutable in the equation
  - Language familiarity worries
    - Trying to hide it rather than learn it (ex. GWT, some Ruby frameworks, some ASP.net frameworks, etc.)
    - Strong type -> Weak type mismatching
- Hard to build a secure system with so many moving parts
- Why can't we use JS server-side?