The Evolution of Server Side Technologies

Sean Thomas

Application Development

S.Thomas14@Unimail.Derby.ac.uk

University Of Derby

Contents

Introduction

Web applications in the present day rely heavily on server side technologies to provide millions of people with their content. Since the very first dynamic websites were created many new languages and server technologies have emerged. The continued growth of the World Wide Web means that the technology that comes with it must grow too. This paper will provide an overlook of where the dynamic web started and how the tools that enable it have evolved.

Birth of the dynamic web

Common Gateway Interface

Before the first scripts were used to generate the dynamic and flexible web page applications that can be seen today, web pages were made from static HTML code which would display fixed content to every user who visited the site. The limited functionality of this method was bound to become more evident as the use of the World Wide Web grew. The Common Gateway Interface was released in 1993 and it enables a way of executing programmes server side. These applications are known as CGI scripts and they supply a more dynamic tool for web development. The scripts are accessed the same way a HTML page is, with a HTTP request. Once a request is received on the server, a new process would be created and the script would run and produce a response. The response would then be sent back to the client browser. Information could now be passed from a client to the server and used within scripts. A very simple code example of this can be seen below written in Perl.

A simple HTML form is shown and asks the user to enter their name and press submit. Once submit is pressed a request is made to run the Form.pl and fed variables from the HTML POST method. The script then receives this data and prints it back to user on a new page.

HTML page

```
<html>
<b>Enter your name</b>
<form action="cgi-bin/ReturnName.pl" method="POST">
<input name="yourname" type="text" size="15"><p>
<input type="submit" value="Click Here">
</html>
```

Perl script

```
#!/usr/bin/perl
use CGI;
my $q = CGI->new();
my $incomingName = $q->param('yourname');
print $q->header;
print "Your Name is: $incomingName";
```

The example above is very simple and does not come close to showcasing the potential CGI created but it does showcase the dynamism of the technology when compared to a Static page. The user can communicate and alter what they see thanks to the CGI script.

CGI at the time afforded developers a choice of what language they would like to write scripts in, as CGI was compatible with any language and server. Certain languages were far more suited to CGI

than others though due to the functionality they provided. As most CGI programming centred on text processing, Perl was a popular choice. The language possesses strong text manipulation features and useful extensions that made database connection easier. Plus it was easy to learn for novice programmers as it was an interpreted language. Perl was quite commonly used for processing form input, which could then be used to search a database. A commercial example could be a page that asks the user to enter a name of a band and the result would display every album released. A simple example can be seen below.

```
#!/usr/bin/perl
use DBI;
use CGI;
my $q = CGI->new();
my $incomingBand = $q->param('bandname');

$db_handle = DBI -> connect("connectionString");

$getAlbum = $db_handle -> prepare ("SELECT album_name FROM albums WHERE
band = '$incomingBand'")

$getAlbum -> execute ;

print $q->header;

while (my $row = $getAlbum->fetchrow)  {
        print "$row\n";
        }
```

The same principle could be used to allow users to register and then log in to a site. The data would be inserted into a database while registering, then when the user wanted to log on their credentials could be checked with the database using Perl's pattern matching abilities.

Other languages such as C and C++ were more suited to experienced programmers due to the fact they required knowledge of memory declarations and type checking. They did not provide the built in advantages of Perl but these could written manually to provide a similar functionality. One of the advantages of using C and C++ was the ability to compile the script into a binary executable, which uses less system resources than an interpreted script and runs faster.

Improving Performance

ISAPI

Although CGI offered many advantages there were some notable drawbacks. The way CGI handled requests caused delay when the site was being visited by a high volume of users. CGI worked by creating a new process for every request it received. This was obviously a big problem in terms of scalability. Newer technology was a must if dynamic websites were to be used with a wider audience.

Microsoft was and still is constantly developing new API, frameworks and plugins for their Internet Information Services (IIS) web server. The Internet Server Application Programming Interface (ISAPI)

is an example of one of these API's. ISAPI was intended to solve the problems faced with CGI, offering some new functionality in the form of extension and filter applications.

ISAPI extensions are written in C++ and compiled into Dynamic Link Library (dll) files, which made them much faster to process when compared with CGI scripts. Another improvement was how the requests were handled. ISAPI uses a multi thread approach instead of running a new process for every request. When a request to run an extension is made, IIS will first check to see if the extension is loaded into the servers memory, if it's not then the extension is loaded into memory. Once that is complete a thread is initialised for the extension. This way the extension is kept in a common address space on IIS so it would be used multiple times where needed, and when memory was needed elsewhere the extension could be unloaded freeing the memory up. Requests on the server could now be processed in larger quantities and use resources in a more efficient manner.

ISAPI filters are also dll files and they are used to improve the functionality provided by the IIS. They are designed to always run in the background of the server, listening to every request until they find one they need to process. Filters could be used for multiple tasks, some of these include handling encryption, performing logging or modifying a request.

Active Server Pages

Using the improved performance of ISAPI extensions, Active Server Pages (ASP) was developed. It was released in 1996 as a server side scripting engine. ASP pages are requested the same way a CGI script is, through a HTTP request. Once an ASP page is requested the Asp.dll ISAPI extension file is executed and it then loads the required scripting language interpreter into the server memory ready to execute any code.

Although using a compiled language delivered benefits to performance, learning how to use C++ for novice programmers was fairly difficult, ASP helped with this. ASP pages were mainly programmed using VBScript or JScript. These languages made ASP a powerful tool, as they had easily readable syntax and provided their own good functionality for programmers of any experience. Knowledge of writing HTML code was enough to get started, as ASP code could be inserted directly into a HTML page using special tags.

```
<%
   Response.Write("ASP says hello!")
%>
```

VBScript came with good support for incorporating Component Object Model (COM) components into scripts and the IIS supplied a few useful objects for developers. One of the more useful objects being the Session object, which provided a great tool for web session management. The session object enables a way of storing information during the lifetime of a user's session. This could be used for sites that allow users to add items to a basket or shopping list, session management provides a way of remembering those items across a whole web session.

On one page the below data could be defined

```
<%
   Session("name")="John"
%>
```

In another the same data could be retrieved with this code

```
Welcome <%Response.Write(Session("name"))%>
```

(W3schools, n.d.)

The Response and Request objects could be used to create and retrieve cookies. These were useful features as cookies could be used to check if a user had visited the site previously, if they had certain details could be remembered on their return to the site. The Server object could be used to create instances of database connections through Activex Data Objects (ADO). The example below using VBScript shows a connection to a database being made, retrieving information from the database and then displaying it. The syntax and objects available make for a more readable piece of code when compared to the same process with Perl CGI.

```
<%
    set conn=Server.CreateObject("ADODB.Connection")
    conn.Provider="Microsoft.Jet.OLEDB.4.0"
    conn.Open "c:/webdata/northwind.mdb"

    set rs=Server.CreateObject("ADODB.recordset")
    rs.Open "Select * from Customers", conn

    for each x in rs.fields
      response.write(x.name)
      response.write(" = ")
      response.write(x.value)
      next
%>
```

(W3schools, n.d.)

Another useful component that could be created from the Server object was the Ad Rotator, which would display a different image each time the page was refreshed or visited. All that is needed is a text file containing information on the adverts. An example is shown below, which demonstrates the simplicity of setting this component up.

```
<%
   set adrotator=server.createobject("MSWC.AdRotator")
   adrotator.GetAdvertisement("Advertisements.txt")
%>
```

Active Server Pages provided programmers with a great tool, though there were some disadvantages. Having the ability to insert ASP code with HTML code had benefits but it could become a problem with bigger projects. In particular this was noticed due to ASP having no debugging features to work with.

A Modern Approach

ASP.NET

By 2002 Microsoft had released the ASP.NET framework as a successor to ASP. It was released with the first version of the .NET framework and enabled many more features and tools to be used for dynamic web development. Updated versions of ASP.NET are still used in the modern day IT industry.

One of the best new features that ASP.NET delivered was that it could be written in any language integrated into the .NET framework, which included C#, C++, VB.NET and many more. The use of any .NET compatible language allowed developers to bring true Object Orientated Programming to web design, a much more manageable and dynamic style of coding for bigger projects. The rise of C# made learning this technology a friendlier task to newcomers. C# is type safe which helps prevent uninitialized variables and going off the end of an array. Programmers did not need to worry about memory declarations either. The fact ASP.NET could be developed in the IDE Visual Studio made this even better. This approach allowed easier use of the code behind model, which is a way of keeping the presentation code and the business logic in a separate class file. It also had the small advantage of being pre compiled before the page was loaded. An example in C# can be seen below.

These statements would exist in the ASP.NET page, which is displayed to the user.

```
<%@ Page Language="c#" AutoEventWireup="false"
Codebehind="ExampleForm.aspx.cs" Inherits="ExampleForm"%>
<asp:Button runat="server" id="btnHello" Text="Hello">
<asp:Label  runat="server" id="lblHello">

 OnClick="btnHello_Click" />
```

This code would exist in the ExampleForm.aspx.cs file declared in the ASP.NET page.

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class ExampleForm : System.Web.UI.Page
{
    protected void BtnHello_Click(object sender, EventArgs e)
    {
        lblHello.Text = "Goodbye!";
    }
}
```

The .NET framework comes with hundreds of built in classes, methods and namespaces for develops to use at their leisure. One of the namespaces available is the System.Data namespace, which contained many useful database features such as ADO.NET. Using ADO.NET, a connection to a database would only be alive while data was either being read or updated and the connection instance could be used for multiple transactions. If reading from a database the data would be copied using XML which could then be accessed through DataSet objects. Within the DataSet object

multiple tables could exist in the form of DataTable objects. This meant larger amounts of data could be read at one time using a single connection instance(ADO HERE). Useful controls built into the framework could be used to display database information; tables could be generated from a control called a GridView using a DataSource.

The example below written in C# shows how to open a database connection, read information from the database and display it with a GridView. The language looks more structured and again makes for easier reading and understanding.

```
private void BindGridView()
{
   using (SqlConnection conn = new SqlConnection("connection String"))
   {
      // Create a DataSet object.
      DataSet dsPerson = new DataSet();

      // Create a SELECT query.
      string strSelectCmd = "SELECT PersonID,LastName,FirstName FROM Person";

      // Create a SqlDataAdapter object
      SqlDataAdapter da = new SqlDataAdapter(strSelectCmd, conn);

      // Open the connection
      conn.Open();


      // Fill the DataTable named "Person" in DataSet with the rows
      // returned by the query.new n
      da.Fill(dsPerson, "Person");

      // Get the DataView from Person DataTable.
      DataView dvPerson = dsPerson.Tables["Person"].DefaultView;

      // Set the sort column and sort order.
      dvPerson.Sort = ViewState["SortExpression"].ToString();


      // Bind the GridView control.
      gvPerson.DataSource = dvPerson;
      gvPerson.DataBind();
   }
}
```

(msdn.microsoft.com, 2017)

Event Driven Programming is another great tool that ASP.NET permits with ease. It gives the developer more control on what can be done with webpage. A form with many buttons can be structured in a convenient way, as code for when each button is clicked can be kept in separate event handling methods. The GridView example above could have its functionality extended with

Event Driven Programming. The output of the gridView would look like a table on a web page, using event driven programming you could trigger an event when a certain cell of the table was clicked. This could be used to create dynamic approach to editing, writing or deleting data from a database.

Real Time Web Applications

Node .JS

One of the more recent additions to the dynamic web is Node.JS (Node) which was first released in 2009. It is programmed using JavaScript and utilises the languages features. One of these features being asynchronous non-blocking input and output processing, which means there is no order to what code is executed. JavaScript has a call back feature which can be used for signalling when a I/O operation has completed. At this point it is pushed onto the Event Loop queue and executed once the preceding call backs have been executed. This process works due to the architecture of Node. Rather than using the multi-threaded request technique that ASP.NET and many others adopt, Node uses a single thread to handle all client requests and an event loop to handle all of the asynchronous tasks. This allows developers more choice in what they'd like to display at any given moment. Multiple requests can be sent from the client and server without the client needing to wait for responses. If used correctly the performance and scalability of Node is impressive. By using a single thread, far less resources are used which allows Node to achieve very high levels of scalability.

An example of how this could be beneficial is the process of downloading a file.

```
downloadPhoto('http://coolcats.com/cat.gif', handlePhoto)

function handlePhoto (error, photo) {
  if (error) console.error('Download error!', error)
  else console.log('Download finished', photo)
}

console.log('Download started')
```

(Callbackhell, n.d.)

This example shows a download photo function with a handlePhoto function as its call back. The handlePhoto function is not called until the downloadPhoto function is complete. This means you can keep your programme running in a non-blocking state without needing to wait for the code to be executed in a line by line manner. The console output at the bottom of the example will be executed before the handlePhoto function can execute. For this reason Node has become a good tool to use for real time applications, such as chat rooms or online gaming apps where many real time events are handled. However the performance of Node JS may not be so good with applications that process large amounts of data frequently.

The production time of developing a project in Node is often shorter, requiring fewer personnel in the process. The same language can be used client side and server side and many programmers already had experience in using JavaScript from client side web design.

One of the setbacks of Node can be due to the call back single threaded architecture, which is prone to struggle under more complex web applications if they are not designed correctly. A term known as call back hell is used for this problem. A main cause of this can be from programmers who have not yet managed a full grasp on how JavaScript works.

Conclusion

Server side programming has come a very long way since CGI and the features provided by Node JS and ASP.NET are clear examples of the strides made. Programmers in the present day have access to a wide range of tools to develop web applications. This enables a decision to be made based on a number of performance factors and the requirements of a project. The advantages Node .JS provides for making real time applications may not provide the same beneficial effect on applications that require heavy use of the processing power. No technology currently leads the line in every area of web development, but the common factor that they all share is they continue to evolve.

References

1. Advantage Of ISAPI Over CGI. (2012). [Blog] *ASP.NET–What is ISAPI ?/What is CGI?/ Advantage Of ISAPI Over CGI*. Available at: https://microsoftmentalist.wordpress.com/2012/01/24/asp-net-advantage-of-isapi-over-cgi/ [Accessed 19 Mar. 2018].

2. Allen, R. (2003). *WindowsDevCenter.com*. [online] Windowsdevcenter.com. Available at: http://www.windowsdevcenter.com/pub/a/network/2003/11/18/activedir_ckbk.html [Accessed 18 Mar. 2018].

3. Callbackhell. (n.d.). *Callback Hell*. [online] Available at: http://callbackhell.com [Accessed 28 Mar. 2018].

4. Capan, T. (2013). *Why The Hell Would I Use Node.js? A Case-by-Case Tutorial*. [online] Toptal Engineering Blog. Available at: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js [Accessed 28 Mar. 2018].

5. Chandrayan, P. (2017). *How Node.Js Single Thread mechanism Work ? Understanding Event Loop in NodeJs*. [online] codeburst. Available at: https://codeburst.io/how-node-js-single-thread-mechanism-work-understanding-event-loop-in-nodejs-230f7440b0ea [Accessed 28 Mar. 2018].

6. Dausinger, M. (2015). *10 weeks of node.js after 10 years of PHP*. [online] Medium. Available at: https://medium.com/unexpected-token/10-weeks-of-node-js-after-10-years-of-php-a352042c0c11 [Accessed 29 Mar. 2018].

7. Dimov, J. (n.d.). *Security Issues in Perl Scripts*. [online] Cgisecurity.com. Available at: http://www.cgisecurity.com/lib/sips.html [Accessed 11 Mar. 2018].

8. En.wikipedia.org. (2017). *Internet Server Application Programming Interface*. [online] Available at: https://en.wikipedia.org/wiki/Internet_Server_Application_Programming_Interface [Accessed 14 Mar. 2018].

9. Garger, J. (2010). *Advantages of Active Server Pages (ASP)*. [online] Bright Hub. Available at: https://www.brighthub.com/computing/hardware/articles/44142.aspx [Accessed 15 Mar. 2018].

10. Gundavaram, S. (1996). *CGI programming on the World Wide Web*. 1st ed. [S.l.]: O'Reilly.

11. msdn.microsoft.com. (2017). *ASP.NET GridView control demo (CSASPNETGridView) in C#, HTML for Visual Studio 2010*. [online] Available at: https://code.msdn.microsoft.com/CSASPNETGridView-5b16ce70 [Accessed 26 Mar. 2018].

12. Msdn.microsoft.com. (n.d.). *How Does ISAPI Compare with CGI?*. [online] Available at: https://msdn.microsoft.com/en-us/library/aa269890(v=vs.60).aspx [Accessed 15 Mar. 2018].

13. Msdn.microsoft.com. (n.d.). *How Does ISAPI Compare with CGI?*. [online] Available at: https://msdn.microsoft.com/en-us/library/aa269890(v=vs.60).aspx [Accessed 28 Mar. 2018].

14. Pingdom Royal. (2007). *A history of the dynamic web*. [online] Available at: https://royal.pingdom.com/2007/12/07/a-history-of-the-dynamic-web/ [Accessed 1 Mar. 2018].

15. Sassenrath, C. (2005). *Creating and Processing Web Forms with CGI (Tutorial)*. [online] Rebol.com. Available at: http://www.rebol.com/docs/cgi2.html [Accessed 5 Mar. 2018].

16. Stevenson, M. (2016). *Perl and the birth of the dynamic web*. [online] Opensource.com. Available at: https://opensource.com/life/16/11/perl-and-birth-dynamic-web [Accessed 3 Mar. 2018].

17. W3schools. (n.d.). *ADO Recordset*. [online] Available at: https://www.w3schools.com/asp/ado_recordset.asp [Accessed 27 Mar. 2018].

18. W3schools. (n.d.). *ASP Session object*. [online] Available at: https://www.w3schools.com/asp/asp_sessions.asp [Accessed 28 Mar. 2018].

19. Weissinger, A. (1999). *ASP in a Nutshell: a desktop quick reference*. 1st ed. [ebook] Sebastopol: O'Reilly, pp.31-38. Available at: https://doc.lagout.org/programmation/ASP.Net/O%27Reilly%20ASP%20In%20a%20Nutshell.pdf [Accessed 18 Mar. 2018].