Scientific Research Publishing

Journal of Computer and Communications, 2019, 7, 1-10
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

The Visualization of Simulated Load-Bearing Soil Particles Using HttpHandler of ASP.NET

Fang Hu

Department of Information Science & Technology, Sichuan Staff University of Science and Technology, Chengdu, China
Email: hf_sc@126.com

How to cite this paper: Hu, F. (2019) The Visualization of Simulated Load-Bearing Soil Particles Using HttpHandler of ASP.NET. *Journal of Computer and Communications*, **7**, 1-10.
https://doi.org/10.4236/jcc.2019.74001

Received: February 24, 2019
Accepted: April 5, 2019
Published: April 8, 2019

Abstract

The Mincrosoft.NET Framework library provides a rich set of components and services for applications, greatly increasing the efficiency of software development. As the HttpHandler features no control parsing, it does not need to go through complicated page processing. Thus, the efficiency is significantly improved. It is especially suitable for the case where the server does not need to return HTML results to the client. This article demonstrates a simple application where HttpHandler and Ajax are combined to transfer large image data to the browser page and keep the display smooth, which proves their efficacy.

Keywords

HttpHandler, Ajax, Asynchronous Request

1. Introduction

ASP.NET is a worldwide prevalent programming framework based on the Microsoft's next-generation. Net platform, utilizes the Common Language Runtime to provide users with powerful enterprise-class Web application services at the server backend. During the Web application development based on ASP.NET, a large number of server controls and services provided by ASP.NET Framework enable us with the convenience as well as the efficiency due to the characteristics of Web applications.

In the simulation of soil particle distribution, the picture will be "flashing" constantly due to the application of webform, and the simulated graph cannot be clearly and smoothly displayed. In this scenario where the execution efficiency is highly emphasized and a large number of images are returned to the client, using webform is not competent. Instead, we could use another web component pro-

vided by ASP.NET: the HttpHandler, which is significantly time-saving and could transfer relatively larger data. And it should be noted that in this article, the application programming and operation are both carried out under IIS 6.0.

2. Page Request and Response in ASP.NET

2.1. IIS Processing

Suppose the browser sends a request to the web server to retrieve the contents of an ASP.NET site. When the request arrives at the web server, it is received by the HTTP.SYS. HTTP.SYS processes the information about this HTTP request and distributes it to the appropriate application pool based on its URL. Once the application pool receives the request and passes it to the worker process (w3wp.exe), w3wp.exe checks the URL suffix of the request to determine the most proper ISAPI extension. The process of selecting the ISAPI extension is called "program mapping", and the ISAPI extension responding to a HTTP request is called a "HTTP Handler" [1] [2].

IIS specifies that the request for the ASP.NET page, whose extension is .aspx, will be handled by aspnet_isapi.dll (for generic handlers, the request with the extension .ashx is also handled by aspnet_isapi.dll). When the worker process (w3wp.exe) loads aspnet_isapi.dll, it will create an HttpRuntime class, which is the entry for the application, and reacts to the request through the ProcessRequest method. Once the ProcessRequest method is called, an instance of the HttpContext is generated, followed by the HttpRuntime loads an HttpApplication object via the HttpApplicationFactory class. Multiple HttpModules are configured in HttpApplication. These HttpModules preprocess the request and then reach the HttpHandler.

All requests go through the HttpModule to the corresponding HttpHandler, and the HttpHandler starts processing the request. At this point, the ASP.NET page life cycle begins. At the end of the processing, results were further modified and returned by HttpHandler to HTTP.SYS. HTTP.SYS stores it in the buffer and forwards the result back to the requesting browser. Eventually, we get the content of the server response. The process flow is illustrated in **Figure 1**.

2.2. The ASP.NET Page Life Cycle

The definition of the ASP.NET page is as follows:

```
public partial class _Default : Page {
        protected void Page_Load(object sender, EventArgs e)
        {     } }
```

Judging from the definition, all pages inherit from the Page class, whose definition is as follows:

public class Page: System.Web.UI.TemplateControl, System.Web.IHttp Handler

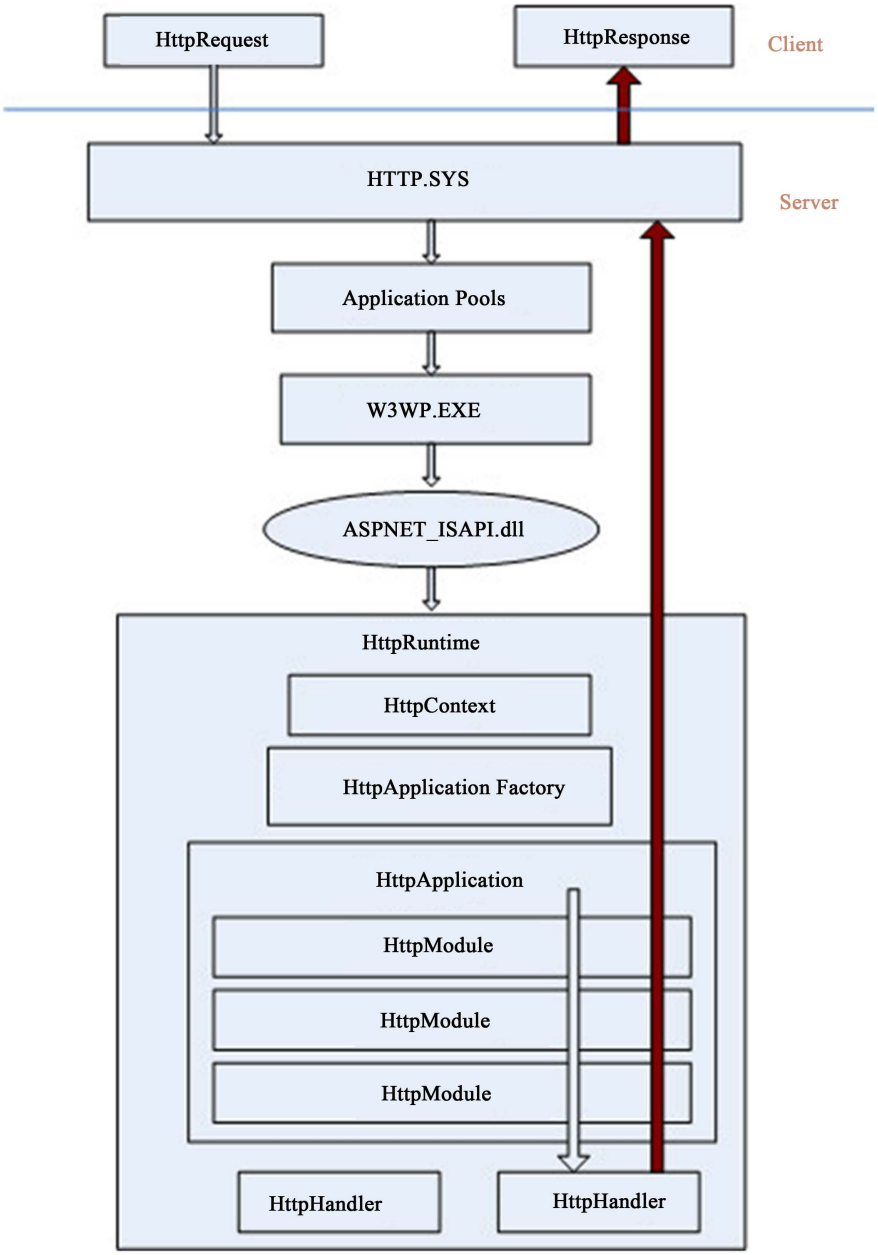


Figure 1. The IIS workflow of processing requests.

The life cycle of a general ASP.Net page goes through the following stages:

1) The initialization of the page framework. This step raises the Init event for this page.

2) Loading the server control onto the Page object.

3) Handling the Postback event defined by the control in this page.

4) Generating corresponding HTML codes and render the Page object.

5) The page is unloaded from memory, triggering the Unload event.

At this point, the browser receives the HTML code of the entire page returned by the server and forwards it to the user. When the page is loading, the events executed from the Page class are summarized in **Table 1** [3] [4].

Table 1. The method execution sequence when the page is loaded.

Sequence	Method	Notes
1	On Pre Init	Triggering Pre Init event in the initialization stage
2	OnInit	Triggering Init event
3	On Init Complete	Triggering Init Complete event once the initialization stage comes to an end
4	On Pre Load	Triggering On Load (Event Args) event at the point when the Post Back Data has been loaded unto the page serve controls yet the PreLoad event has not been activated
5	Page_Load	
6	On Load	Triggering Load event
7	Button_Click	Triggering Postback event
8	On Load Complete	Triggering Load Complete event when the page loading has been accomplished
9	On Pre Render	Triggering Pre Render event
10	Pre Render Complete	Triggering On Pre Render Complete event after the Pre Render Complete event yet the final page has not been presented
11	On Save State Complete	Triggering Save State Complete event after the page state has been restored
12	On Unload	Triggering Unload event

3. The HttpHandler

The HttpHandler is the fundamental core of the ASP.NETAs one of the .NET Web components. It is a class that contains the IHttpHandler special interface. Any class that contains this interface can be used as a target program for the external request.

Generally, the HttpHandler could match classes in terms of most tasks. The workflow is as follows: a HttpHandler will obtain the client's request submitted to the server, accessing the server's file system and database, and form a non-HTML response content which will be returned to the client after the processing is completed. The definition of a HttpHandler is illustrated as follows:

```
public class Handler1 : IHttpHandler {
        public void ProcessRequest(HttpContext context)
        {    ……
             context.Response.ContentType = "text/plain";
             context.Response.Write("…."); }
        public bool IsReusable{
             get
        {return false; }
        }
    }
```

Several explanations are presented here for better comprehension:

Property IsReusable: Acquiring a value indicating whether another request can use an IHttphandler instance.

Method ProcessRequest: Processing request using this method.

Class HttpContext provides access to the internal Request, Response, and Server properties of the request.

4. Asynchronous Javascript and XML

As stated before, when any part of the web page needs updating, the entire page will be automatically overloaded. Ajax (Asynchronous JavaScript and XML) enables asynchronous updates of web pages by performing small amounts of data exchange with the server backstage. This feature allows users to update the local content of a web page without reloading the entire web page.

The core of Ajax technology is the XMLHttpRequest object, which offers a way to communicate with the server even after the page is loaded. Some functions include:

1) Updating the page without reloading the page;

2) Requesting data from the server after the page has been loaded;

3) Receiving data from the server after the page has been loaded;

4) Sending data to the server backstage.

Some further properties are introduced as follows.

1) The major properties of the XMLHttpRequest object are explained as follows:

readyState: state description which displays the state of the XMLHttpRequest. Its value changes from 0 to 4 and corresponds to different states:

0: Request not initialized

1: Server connection is established

2: Request has been received

3: Request is in processing

4: The request has been completed and the response is ready. The data can be retrieved via the corresponding attribute of the XMLHttpRequest object.

Status: Status attribute contributes to returning the server status code. When it equals 200, it represents the situation that the server has successfully accepted the client request.

Event handle on ready state change: The event handle on ready state change will be activated when the state of the XMLHttpRequest object changes. Its value is the name of this function.

responseText: Returns a response as a string.

The methods provided by the XMLHttpRequest object:

Open(method, url, async): method corresponds to request type, which is POST or GET; url corresponds to the location of the file on the server; async corresponds to true (asynchronous) or false (synchronous).

Send(): Send the request to the server.

2) The basic working flow of AJAX is shown in **Figure 2**.

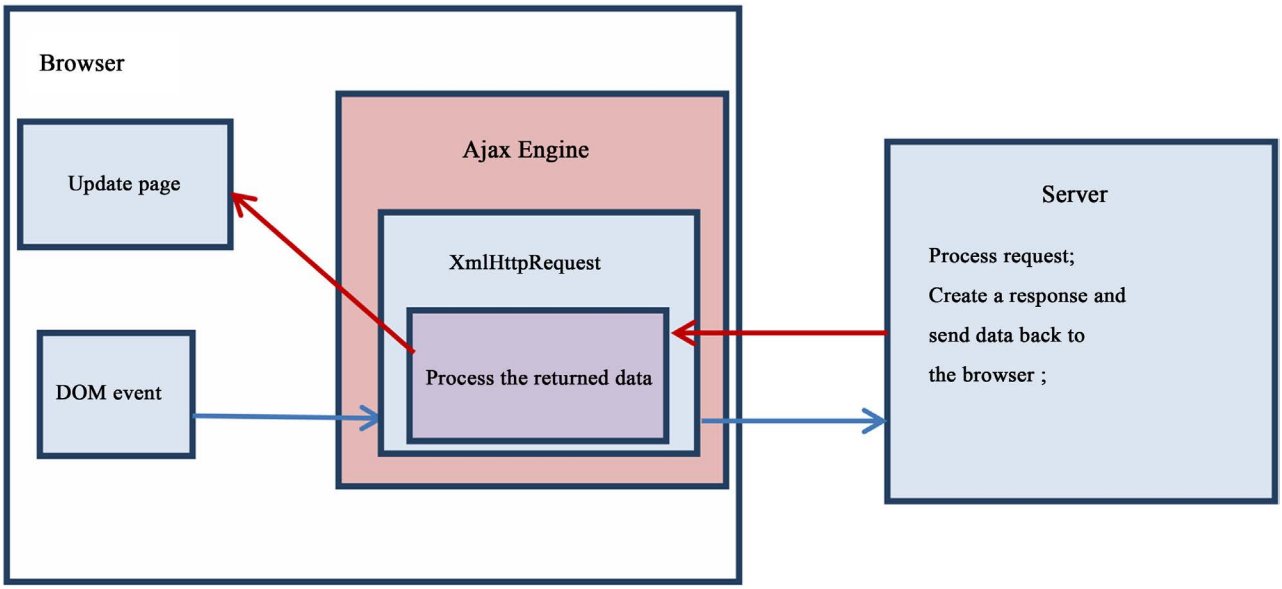


Figure 2. The basic working flow of AJAX.

3) JSON (JavaScript Object Notation)

JSON is a syntax for storing and exchanging text information. It is a lightweight text data-interchange format that is easy for people to read and write, which is also true for machine parsing and generation. It is smaller and faster than XML and uses the built-in JavaScript eval() method for parsing. AJAX technology makes each request faster and returns only the data it needs for each request. Overall speaking, for AJAX applications, JSON is faster and easier than XML [5] [6].

5. Application

In the discrete element modelling, which is a kind of numerical simulation especially suitable for soil mechanics research, it simulates soil as particle assemblage and the soil is assumed to be composed of quantities of small particles. In order to conduct more comprehensive qualitative and quantitative analysis, it is often necessary to smoothly display the information such as the motion trajectory, force chain and velocity of the soil particles during the loading process. This application works as follows. At the application server back end, based on the calculated response of the sample data under different conditions and under different stress states, we continuously generate a large number of soil sample particle distribution maps and present them on the browser front end.

This application needs to continuously obtain the particle distribution map from the server, display it in the browser, and present various types of simulated motion trajectories. The front end adopts Ajax asynchronous request to avoid the "flashing" of the picture due to the refresh of the whole page when the picture is displayed on the screen; the back end is processing efficiently drawing upon the general processing program. Together, the combination of these two parts realizes the continuous and smooth illustration of the graphic.

Basic front-end code is as follows:

```
…
<script type ="text/javascript" >
        var request = new XMLHttpRequest();
        var intplay1=setInterval("showpic ()",500);
        var data="id=0";
    function showpic (){
     request.onreadystatechange = state_Change;
     request.open("GET", " Handler1.ashx?"    + data, true);
     request.send(null);
     }
          function restart(){
               intplay1=setInterval("showpic ()",500); }
          function state_Change() {
               if (request.readyState == 4) {// 4 : "loaded"
                    if (request.status == 200) {// 200 : "OK"
                         var data = eval('('+ request.responseText + ')');
                        document.getElementById('Image1').src = unescape (data.addr);
                    }
                    else {    ..... }    }
               else { …… }
          }
</script>
…
```

1) The client sends a request to the server every 500 milliseconds.

2) Instantiate the XMLHttpRequest object:var request = new XMLHttpRequest();

3) Send the request to the server:request.open(“GET”, “Handler1.ashx?” + data, true);

request.send(null); where the parameter true means that the request is processed asynchronously, and it could continue without waiting. When the response is ready, the response is processed.

4) When the readyState property changes, the on ready state change event is triggered, and the state_Change() response function is executed. When the request is finished, the information returned from the server is obtained. The key statement in the response function state_Change() is as follows:

var data = eval('('+ request .responseText + ')'); document.getElementById ('Image1').src = unescape (data.addr);

Where request.responseText is used to set the response data obtained from the server as a string.

After utilizing the eval() method to convert the JSON string returned from the server into a JSON object and assigning the image address in the JSON object to the page picture control, the image is displayed in the control.

Basic back-end code is as follows:

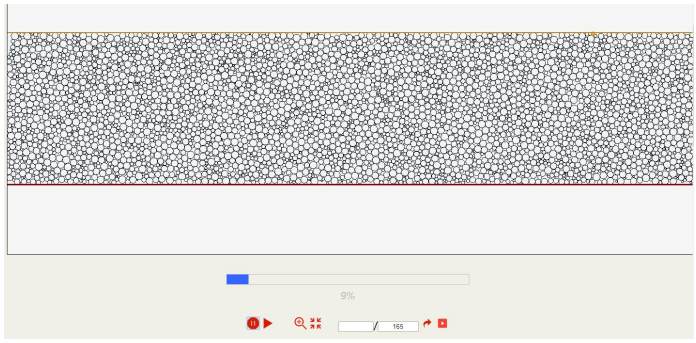
```
public class Handler1 : System.Web.IHttpHandler{
    private string sss; public static int stack = 0; public static int stack0;
    public void ProcessRequest(HttpContext context)
    {    string addr = "";    StringBuilder sb = new StringBuilder("{");
         sss = context.Server.MapPath("~/images/");
         context.Response.ContentType = "text/plain";
         int i;
         string[] returnValue;
         returnValue = Directory.GetFiles(sss);
         string[] getName = new string[returnValue.Length + 1];
         string iid = "0";
         iid = context.Request.QueryString["id"].ToString();
         if ((Convert.ToInt32(iid) > 0))
              stack = Convert.ToInt32(iid);
         stack0 = returnValue.Length;
         if (returnValue.Length > 0) {
           for (i = 0; i <= returnValue.Length - 1; i++) {
                 if ((stack == i)) {
                      getName[i] = System.IO.Path.GetFileName(returnValue[i]);
                      addr = "images/" + getName[i]; iid = Convert.ToString(stack);
                      sb.Append("id:" + iid); sb.Append(",addr:'" + addr + "'");
                      sb.Append(",all:" + Convert.ToString(stack0));    sb.Append("}");
                    ...
                      context.Response.Write(sb.ToString());
                      context.Response.End();    break;
                 }    }
          ... }   }

    public bool IsReusable { get { return false; } }}
```

The server receives the request, executes Handler1.ashx, and processes the received Http request via the process Request method where it accesses the server file system, obtains the image information, and constructs a JSON string. The string format is: id:value1, addr:value2, all:value3. The server then returns the JSON string which contains the order that images are stored in the server, the image storage path, the total number of images, and so on.

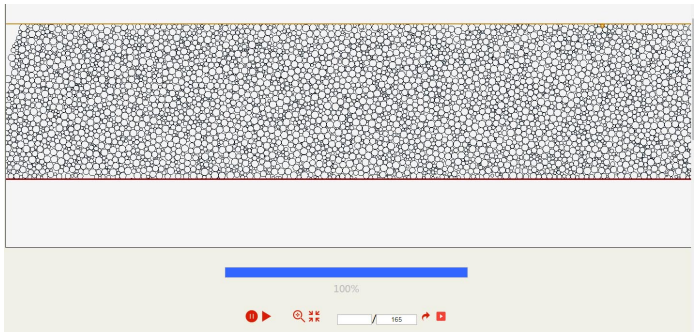
Snapshots of results are shown in **Figure 3** and **Figure 4**.

6. Conclusion

In summary, we use HttpHandler in the Server and AJAX technology in the Client to display a large number of images smoothly and intuitively, which makes us intuitively observe the simulated trajectory of soil particles. As a very simple, highly efficient and cost-effective tool, the HttpHandler, combined with AJAX technology, could play a significant role in handling requests for non-HTML result return. It could be widely utilized in processing data such as RSS, feeds, images and files from the server.

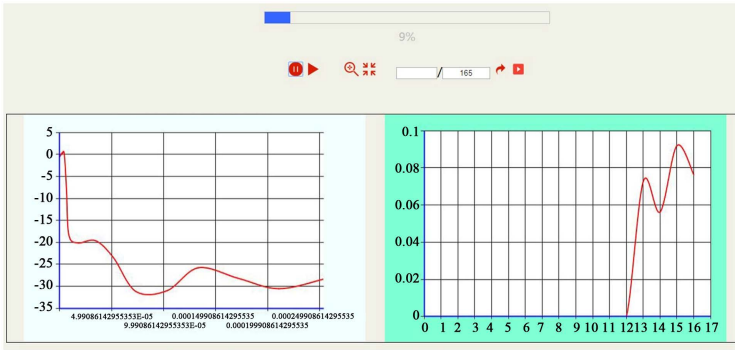


(a)

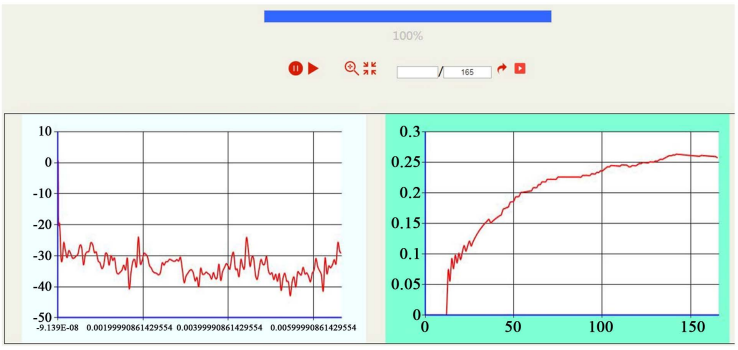


(b)

Figure 3. The result of force chain demonstration. Two pictures show different stage indicated by the progress bar.



(a)



(b)

Figure 4. The result of curves demonstration. Two pictures show different stage indicated by the progress bar.

Acknowledgements

This work is supported by Department of Information Science & Technology, Sichuan Staff University of Science and Technology.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Yang, S. (2012) Page Staticization Based on HttpHandler. *Digital Technology and Application*, No. 1, 50-51. (In Chinese)

[2] Paulson, L.D. (2005) Building Rich Web Applications with Ajax. *Computer*, **38**, 14-17.

[3] Mesbah, A. and Van Deursen, A. (2009) Invariant-Based Automatic Testing of AJAX User Interfaces. 2009 *IEEE* 31*st International Conference on Software Engineering*, Vancouver, BC, 16-24 May 2009, 210-220. https://doi.org/10.1109/ICSE.2009.5070522

[4] Galhardas, H., Florescu, D., Shasha, D. and Simon, E. (2000) AJAX: An Extensible Data Cleaning Tool. *ACM Sigmod Record*, **29**, 590. https://doi.org/10.1145/335191.336568

[5] Mesbah, A. and Van Deursen, A. (2007) Migrating Multi-Page Web Applications to Single-Page Ajax Interfaces. 11*th European Conference on Software Maintenance and Reengineering* (*CSMR*'07), Amsterdam, 21-23 March 2007, 181-190. https://doi.org/10.1109/CSMR.2007.33

[6] Qun, Y. and Zhang, J.B. (2014) Design of Cloud Services Platform Based on JSON. 2014 9*th International Conference on Computer Science & Education*, Vancouver, BC, 22-24 August 2014, 560-565.