

web

by Zeyi Wan

FILE	17594_ZEYI_WAN_WEB_242847_1545689718.DOCX (73.92K)		
TIME SUBMITTED	09-APR-2020 12:21AM (UTC+0100)	WORD COUNT	3269
SUBMISSION ID	123824263	CHARACTER COUNT	19567

Web Development

Name: Zeyi Wan

Student Number: 1904718

Content

1 Server-side code and Model View Controller design pattern	1
1.1 Server-Side Code	1
1.2 Model View Controller design pattern	1
1.2.1 What is Model View Controller	1
1.2.2 Why use MVC design pattern	2
2 Compare and contrast client-side and server-side scripting	3
3 Node.js	7
3.1 What is Node.js	7
3.2 Pros and Cons of Node.js	8
3.2.1 Why use Node.js	8
3.2.2 Some disadvantages of node.js	9
4 Ajax	10
4.1 Why Ajax is important	10
4.2 Examples of Ajax applications in page development	11
Reference	17

1 Server-side code and Model View Controller design pattern

1.1 Server-Side Code

For general purposes, server-side processing is primarily used to interact with permanent storage such as databases or files or to process user input, etc. [1], while server-side code is the programming code used for the interactions that run on the server. As opposed to client-side code, the main function of server-side code is to analyze and process the operations and data submitted by the user, and then return the processing results to the client, after which the server-side code does not continue to participate in the analysis and display of the returned results. For example, when a customer registers a platform user, the operation of case switching is run on the user's client and has nothing to do with the server side, but once the registration form is submitted to the server side, the server-side code will parse the submitted data and store it in the database, the operation is successful and return the success information to the client, after that, how to parse and display the success information to the user will be carried out by the client-side code.

1.2 Model View Controller design pattern

1.2.1 What is Model View Controller

The MVC (Model View Controller) pattern is a software architecture patterns that is now widely used in software development and was first proposed by Trygve Reenskaug in 1978 [2].

The architecture pattern includes three basic parts, Model, View and Controller, as shown in the MVC architecture section. It is mandatory to keep the input, processing, and output of the application separate and is generally used to implement reusable and dynamic programming [2]. Each basic part can be reused with minor modifications or

extensions, making the program structure simpler and more intuitive, improving development efficiency and greatly reducing development costs. Among them, the Model part is mainly responsible for implementing business functions, encapsulating the core data, logical relationships and business rules of the application problem, and providing the processing process of business logic; the View part does not have any logic code, but only describes how to render part of the Model content as a visible interface and display it to the user. To the user, the View is a representation of the efficiency of the application, representing the application provided to the user, while providing the basis for the first critique of the application and reference to the model, and converting the model into a format ready to be provided to the user; the Controller part is responsible for processing requests, performing operations on the Model and selecting a page to return to the user. MVC-based applications are mainly implemented through the interaction between these three parts to achieve their various functions.

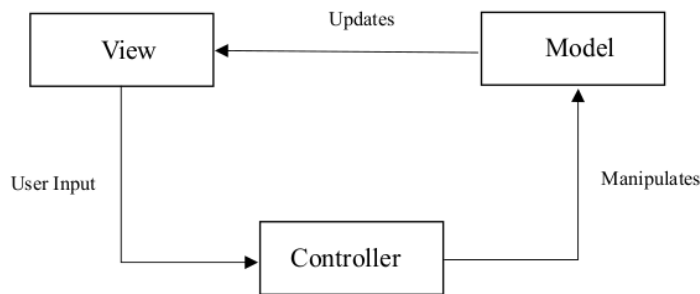


Figure 1 Structure of the MVC pattern

1.2.2 Why use MVC design pattern

➤ Low Coupling

Due to the inherent nature of the MVC design pattern, the View and Model layers are separated from each other, which allows developers to update the View layer code alone without changing the Model layer code, and if the business logic changes, developers can simply change the Model layer code.

➤ Highly Reusable

The MVC design pattern allows multiple Views of different styles to access the same Model, simply by processing the data format in the Controller, without modifying the Model layer code. For example, when shopping online, each product corresponds to a different View, but when the consumer places an order, the business logic for processing the order is the same, so it can all be forwarded by the Controller to the same Model for processing.

➤ Maintainability

Since MVC separates software development into Model, View and Controller modules, the separation of View and Model in particular makes web applications easier to maintain and modify. For example, developers only need to change the module that want to change without worrying about changes to other modules.

➤ Quick Deployment

The MVC design model greatly shortens the software development cycle, because the system is divided into several parts, and the developers of each module have their own roles, then the division of labor in development can be better realized. Web designers can develop the JSP in the View layer, developers familiar with the business can develop the Model layer, and other developers can develop the Controller layer, while reducing communication costs.

➤ Software Engineering Management

The various modules of MVC may contain the same features, based on which the same points can be easily managed through software tools. At the same time, because of the existence of the Controller, developers can use the Controller to combine different Views and Models to achieve different business functions.

2 Compare and contrast client-side and server-side scripting

Client-side and server-side scripting programs can be written in the same scripting language. But the way in which client-side scripting is written differs greatly from server-side scripting, mainly because of the differences in the services available in the

two different environments.

In the client environment the scripting program is run in the browser. A browser is a program that displays HTML or other objects to users of a web site. Thus, in the browser, the services available to client-side scripts include the ability to view history (history is a list of URLs that have been accessed), the ability to view different elements of an HTML document (such as form elements or hyperlinks to HTML text), and the ability to humanize other HTML documents on Internet into the browser.

The server-side scripting program is run on a web server. Thus server-side scripts have completely different services available. These services include determining the type of browser used and the functionality of that browser (e.g., whether it supports ActiveX controls or Java applets), etc.

The following tasks are usually done by client-side scripts:

- (1) Generate a custom HTML when the page is loaded into the browser.
- (2) Validate the data from the keyman to the HTML form before sending it to the server.
- (3) Display information in response to user events.

The following tasks are usually done by server-side scripts:

- (1) Database connection.
- (2) Search the database, send the results to an HTML page.
- (3) Validation of the data in the database.
- (4) Under certain conditions, the server is redirected to a new page.
- (5) Processing user keyman information into HTML forms.
- (6) Browser detection.
- (7) Counter.

The scripting service available to a script is provided by a built-in object. Services can be obtained by means of methods and properties of built-in objects. Because services

are runtime dependent (client-side services are completely different from server-side services), the built-in objects available for client-side scripts and server-side scripts are also completely different. Fortunately, client scripts are designed with standardization in mind. The built-in objects available for client-side scripts are the same regardless of the scripting language used by the developer (such as VBScript, JavaScript, Perl, or other scripting languages). They are Window object, Document object, Navigator object, History object, Link object, Anchor object, Location object, Form object. For example, a client-side script can make a reference to a Window object. The window object within a script represents the current window or a text box within a document to which the script is embedded. With this window object, the scripter can change the HTML document it is currently viewing, view different form elements in the current HTML document, or view the history of URLs visited in the current window.

The server-side scripts, on the other hand, have completely different built-in objects. These built-in objects include Application objects, Session objects, Request objects, Response objects, Server objects, Object-Context objects, etc. When the user submits a Form to the ASP page, the Request object will send data to the ASP program, the program processing results from the Response object back to the browser, both objects exist on the server side.

Here is an example of a client-side script, a server-side script and its built-in objects:
Suppose there is an html file on the server:

```
<html lang="en">
<head>
<title>client</title>
<script type="text/javascript">
    alert("This is client-side scripting");
    function a(){
        alert("client-defined function");
    }

```

```

</script>
<%
    Response.Write ("This is server-side scripting");
%>
</head>
<body>
    <form method="POST" action = "eg.asp">
        Name: <input type="text" name="username"/>
        Age: <input type="text" name="age"/>
        <input type="submit" value="Submit" name="submit"/>
    </form>
</body>
</html>

```

At this point, after the browser loads the page, a client-side scripting dialog will pop up.

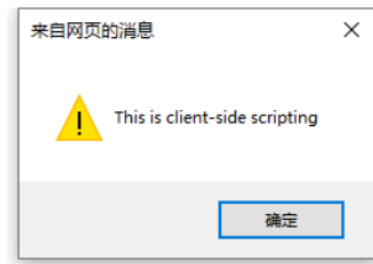


Figure2 client-side scripting dialog

For server-side scripts, since the script is designed to run on the server, it will be displayed directly on the page if the user's browser is unable to resolve the server-side script.

```

<% Response.Write("This is server-side scripting"); %>
Name:  Age:  

```

Figure3 User's browser is unable to resolve the server-side script

If the user submits the form and the server returns the name, the eg.asp on the server is as follows:

```
<html lang="en">
  <head>
    <title>server</title>
  </head>
  <body>
    %<i>xname = request.form("username")%
    returns the name: %<i>response.write xname %
  </body>
</html>
```

A server-side script cannot legally reference the current window object because it does not exist on the web server (there is no visual interface associated with the object in the server environment, so history or the current URL will not be visible). On the other hand, in a server environment, the Request object retains information about the URL of the user request, which can be used to call server-side scripts.

3 Node.js

3.1 What is Node.js

In general, Node.js is the underlying environment in which JavaScript can run on the server, and its open source and cross-platform nature makes node.js widely used [3, 4]. Node.js is similar to the overall structure of java, java needs Java Virtual Machine (JVM) support to run java programs, the cross-platform nature of Java is because no matter what platform, the running environment is the same, and Node.js is also the same. javascript language does not have the necessary file reading and writing ability as a back-end language, but the browser kernel has this function, using JavaScript to call some of the browser kernel as a back-end language necessary function. Node.js can not only be used for webserver writing, but also can do many things that java can do.

3.2 Pros and Cons of Node.js

3.2.1 Why use Node.js

➤ Asynchronous event drivers

Node.js uses asynchronous event drivers, and although node.js uses multiple threads to handle file and network events, node.js applications are running single-process threads, so they take up less server resources. And although it is a single process, multiple instances can be opened via the official cluster module to take full advantage of the multi-core CPU. Saving server resources and achieving the desired state at the same time.

➤ Non-blocking IO [5]

Asynchronous IO refers to the ability of the operating system to provide IO (data in and data out) that behaves inconsistently at different OS levels. There is asynchronous non-blocking; there is synchronous blocking, but in any case it can be seen as data interaction between the upper layer applied to the lower layer of the system; the upper layer is dependent on the lower layer, but in turn the upper layer can modify these capabilities provided by the lower layer; if this operation is asynchronous, non-blocking, then this is asynchronous non-blocking asynchronous IO model.

Since node.js is single-threaded, the server does not need to wait for the return of asynchronous operation results and can continue to execute the program directly. Until the asynchronous operation results are returned, the asynchronous event is triggered, and the main thread is notified, and the main thread executes the call to the corresponding function code. When I/O operations are extremely intensive, this mechanism will significantly reduce the wait time for business execution.

➤ Lightweight and efficient

The node.js installation package is lightweight and easy to install, allowing for immediate production development without the need for cumbersome environment configuration. And development with node.js is completely unrestricted, allowing

developers to build their ideal software.

➤ Low learning costs

Since the client-side code that interacts with the Node proxy server is written in javascript, both client-side and server-side are written in the same language, which breaks the barrier between the traditional front and back end, developers mastering JavaScript is equivalent to mastering Node.js, learning migration costs are extremely low.

3.2.2 Some disadvantages of node.js

➤ Not suitable for CPU-intensive applications and logically complex operations.

As mentioned above, Node.js uses a single-process thread, although this mechanism takes up less resources, but also brings the disadvantage of not being suitable for CPU-intensive applications, which usually run longer calculations, which will result in CPU time slices not being released for a long time, making it impossible for subsequent I/O to continue to be initiated, causing blockage of the entire program.

At the same time, the control flow of node.js does not conform to people's conventional linear thinking, but is split by a number of different small events, so when developers need to use node.js to implement a logically complex business, not only will development efficiency will be extremely low, and development code is too complex to maintain. So node.js is better suited for logically simple business

➤ Low reliability

Since Node.js uses single-process threads and only supports single-core CPUs, if the developer cannot make full use of code robustness or other ancillary modules through their own development experience, then when the only process in node.js crashes, the entire web service will also crash directly and cannot continue to provide service to users.

➤ More development times [6]

When the developer team is not that experienced with node.js development, then the unopinionated feature of node.js will be one of the reasons for slowing down development. Because after using node.js, developers basically need to write everything about the program from scratch, which greatly reduces productivity.

➤ Immature tool support [6]

Although node.js is currently stable enough, it is not yet mature enough to support the tools associated with Java, so it is not yet suitable for enterprises seeking stability.

4 Ajax

4.1 Why Ajax is important

Ajax is currently in the browser through the JavaScript script can be used by a collection of all technologies . Ajax does not create a specific new technology , it uses are already existing and more mature technology , Ajax through a new way to integrate these technologies together . Specifically, the following technologies are included:

- (1) XHTML: Extensible Hypertext Markup Language.
- (2) CSS: Cascading style sheet, which is responsible for the presentation of client-side page styles.
- (3) DOM: Document Object Model, a common method of representing and processing an HT-ML or XML document.
- (4) Java Script: a mature client-side scripting language , Java Script by manipulating the DOM to change the front-page structure .
- (5) XML: Extensible Markup Language, a language transfer format for Ajax.

The core of Ajax technology is to provide a special object, namely XMLHttpRequest, to communicate with the server asynchronously, without the overall page refresh, thus reducing the amount of data transfer, improve the responsiveness of the web application, bring a good experience to the user. For example: data tables, tree controls, and other complex controls Intelligent client-side controls can communicate with the server via

XMLHttpRequest objects, obtain data, and cache and process data. This allows many tasks to be done on the client side, such as data validation, form validation, etc. Ajax technology has greatly improved the usability of web applications and the user interaction experience, and has ultimately been widely recognized by users and the market.

4.2 Examples of Ajax applications in page development

Due to Ajax's features, two drop-down boxes can be linked in real time without refreshing the page. When the user selects an option in the first drop-down box, the second drop-down box will update the options in real time based on what was selected in the first drop-down box. An example of Ajax combined with a servlet is as follows.

First, the JSP code is as follows.

```
<html>
<head>
<title> Examples of Ajax applications </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<SCRIPT type="text/javascript">

    var req;
    window.onload=function(){

    }

    function Change_Select(){
        var zhi=document.getElementById('class').value;
        var url="select?id="+escape(zhi);
        if(window.XMLHttpRequest) {
            req=new XMLHttpRequest();
        }else if(window.ActiveXObject) {
            req=new ActiveXObject("Microsoft.XMLHTTP");
        }
    }
}
```

```

    }
    if(req) {
        req.open("GET",url,true);
        req.onreadystatechange=callback;
        req.send(null);
    }
}

function callback(){
    if(req.readyState == 4) {
        if(req.status == 200) {
            parseMessage();
        }else{
            alert("error");
        }
    }
}

function parseMessage() {
    var xmlDoc=req.responseXML.documentElement;
    var xSel=xmlDoc.getElementsByTagName('select');
    var select_root=document.getElementById('item');
    select_root.options.length=0;

    for(var i=0;i<xSel.length;i++){
        var xValue=xSel[i].childNodes[0].firstChild.nodeValue;
        var xText=xSel[i].childNodes[1].firstChild.nodeValue;
        var option=new Option(xText,xValue);
        try{
            select_root.add(option);

```

```

        }catch(e){
        }
    }
}
</SCRIPT>
</head>

<body>
    <div align="center">
        <form name="form1" method="post" action="">
            <SELECT name="class" id="class" onChange="Change_Select()">
                <OPTION value="0">==Please Select==</OPTION>
                <OPTION value="1">Vegetable</OPTION>
                <OPTION value="2">Fruit</OPTION>
            </SELECT>
            <SELECT name="item" id="item">
                <OPTION value="0">==Please Select==</OPTION>
            </SELECT>
        </form>
    </div>
</body>
</html>

```

The back-end Servlet code is as follows.

```

public class SelectServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/xml");
        response.setHeader("Cache-Control","no-cache");
    }
}

```

```

        String targetId=request.getParameter("id").toString();
        String xml_start="<selects>";
        String xml_end="</selects>";
        String xml="";
        if(targetId.equalsIgnoreCase("0")){
            xml="<select><value>0</value><text>==Please
Select==</text></select>";
        }else if(targetId.equalsIgnoreCase("1")){
            xml="<select><value>1</value><text>Spinach</text></select>";
            xml
            +="<select><value>2</value><text>Lettuce</text></select>";
            xml
            +="<select><value>3</value><text>Cabbage</text></select>";
        }else if(targetId.equalsIgnoreCase("2")){
            xml="<select><value>1</value><text>Grapes</text></select>";
            xml += "<select><value>2</value><text>Apple</text></select>";
            xml
            +="<select><value>3</value><text>Banana</text></select>";
            xml
            +="<select><value>4</value><text>Cherry</text></select>";
        }
        String last_xml=xml_start+xml+xml_end;
        response.getWriter().write(last_xml);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request,response);
    }
}

```

When the option in the first drop-down box is changed, the Change_Select function is triggered, which takes the selected option id and encapsulates it in Ajax's core object - XMLHttpRequest. When the backend server gets the object, it takes out the option ID and looks for the corresponding secondary option in the database based on that option ID. The backend will then encapsulate the query results and send to the front page to trigger the callback function, if the backend query is successful will call the parseMessage function, the function will iterate through the query results and display them as drop-down options one by one in the secondary drop-down box. The results are shown below.

There are two options in the first level drop-down box.

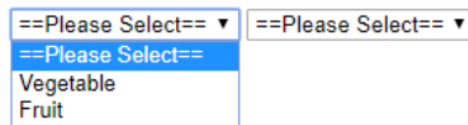


Figure 4 Options in the first level drop-down box

When the user checks the “Vegetable” option, the second level drop-down box will update the relevant sub-options in real time.

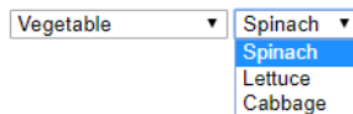


Figure 5 Secondary drop-down box options

When the user changes the first level drop-down box option, the second level drop-down box option will also be available in real time.

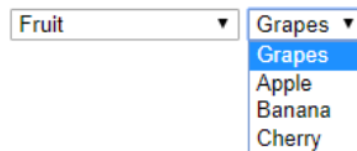


Figure 6 Second level drop-down box options updated in real time

This implementation can make the user experience more smooth, when the user first selects the first level drop-down box options, the front page is equivalent to do a no refresh filtering operation, users do not have to wait for the page to refresh again, but also more convenient and quick to select the second level drop-down box options.

Reference

- [1]"Client-Side vs. Server-Side Code: What's the Difference?", Segue Technologies, 2020. [Online]. Available: <https://www.seguetech.com/client-server-side-code/>. [Accessed: 07- Apr- 2020].
- [2]"Model–view–controller", En.wikipedia.org, 2020. [Online]. Available: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>. [Accessed: 07- Apr- 2020].
- [3]"The definitive Node.js handbook", freeCodeCamp.org, 2020. [Online]. Available: <https://www.freecodecamp.org/news/the-definitive-node-js-handbook-6912378afc6e/>. [Accessed: 07- Apr- 2020].
- [4]"Node.js", En.wikipedia.org, 2020. [Online]. Available: <https://en.wikipedia.org/wiki/Node.js>. [Accessed: 07- Apr- 2020].
- [5]"Introduction to Node.js", Introduction to Node.js, 2020. [Online]. Available: <https://nodejs.dev/>. [Accessed: 07- Apr- 2020].
- [6]"Why to Use Node.js: Pros and Cons of Choosing Node.js for Back-end Development", Netguru.com, 2020. [Online]. Available: <https://www.netguru.com/blog/pros-cons-use-node.js-backend>. [Accessed: 08- Apr- 2020].
- [7]"Ajax (programming)", En.wikipedia.org, 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Ajax_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming)). [Accessed: 08- Apr- 2020].

FINAL GRADE

86/100

GENERAL COMMENTS

Instructor

Server-side/MVC (25) : 19

Client/Server Scripting (25): 24

Node.js (25): 25

AJAX (25): 18

Feedback:

=====

I would like to have seen a little more analysis of the benefits of this approach also the implications for developers e.g. convention over configuration

Both the explanation of scripting and node.js are very good.

Ajax code was a little short, I was hoping for slightly better examples.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19
