HTML1st—A Lightweight Dynamic Web

by

Boyang Tang

Supervisor: Dr Sathiamoorthy Manoharan

BTech 451 Mid-Report

Tamaki Campus
Department of Computer Science
The University of Auckland
New Zealand

June 2016

Abstract

The BTech 451 course is one-year term project of BTech (Information Technology) degree, where the four year degree offered by Department of Computer Science and taught at University Of Auckland. This degree covers not only Computer Science and Information System papers but also contains some business and management courses. As the most important course of this degree, BTech 451 occupies 45 points of full year credits and it is split up into two parts, BTech 451A (15 points) for first semester and BTech 451B (30 points) for second semester. My project is intended to build a light-weight C# engine that handles Server-side Scripting, the engine will convert HTML page embedded C# code fragments to pure HTML page. This middle year report will illustrate what tasks I have done and what problems still exist, also summarize the findings during actions taken through the first part of course, merging cells step by step into a conclusion, meanwhile, give an expected plan of next semester.

Acknowledgments

I would like to express my very great appreciation to my academic supervisor Dr. S. Manoharan for his patient guidance and encouragement. I would also like to thank all those who provide valuable and constructive suggestions in completing the project and writing this report. Last but not least, many thanks to my parents for their supports.

Boyang Tang
Auckland
June 6, 2016

Contents

Abstract ii
Acknowledgements iii

Contents iv

1 Introduction 1
1.1 Project Overview. 1
1.2 Expected Outcomes 2
1.3 Report Structure. 3

2 Project Basics 5
2.1 Programming Knowledge Overview 5
2.1.1 C# .NET. 5
2.1.2 .NET Framework 6
2.1.3 Difference between Framework and Library 8
2.2 Reflection and Dynamically Loading 10
2.2.1 Reflection in .NET 10
2.2.2 Dynamically Loading Assembly 13

3 HTML1st Design 15
3.1 Scripting Language. 15
3.1.1 Scripting Language Overview. 15
3.1.2 Client-Side Scripting. 17
3.1.3 Server-Side Scripting. 18
3.2 Treating C# like A Scripting Language. 21
3.3 HTML Parser. 24
3.3.1 HTML Agility Pack. 24
3.3.2 Processing Instruction 26

4 Conclusion 28

Chapter 1

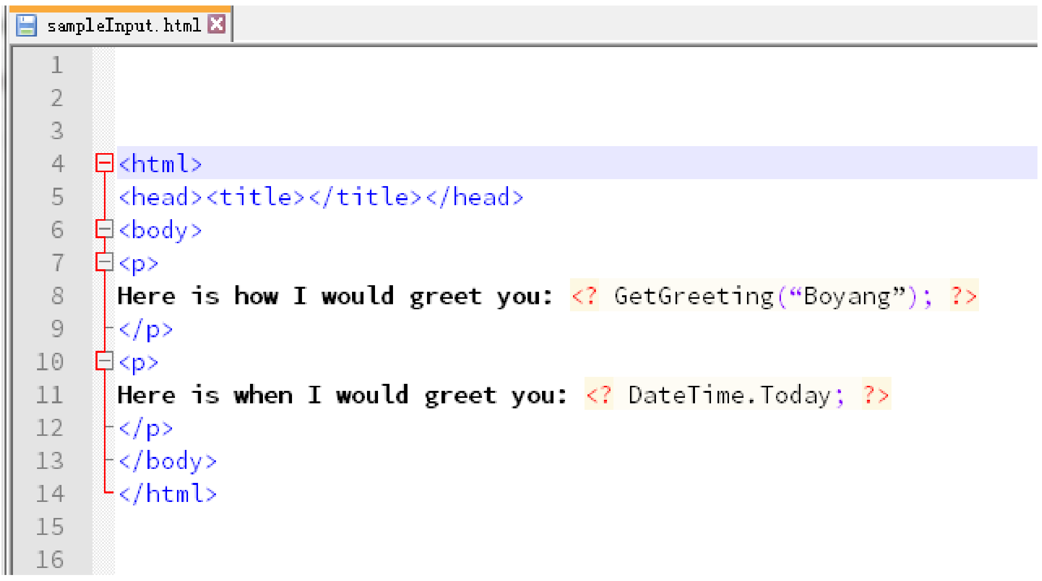
Introduction

This section will give an overall introduction of my project, it illustrates the goal that my project is designed for, and what do I hope to gain from this project, and then conclude the report pattern.

1.1 Project Overview

As so far, the html parser has been widely used in collecting specific data from the intended HTML sources, where these data would motivate the subsequent development activities. With the rise in requirements of acquiring dynamic information, it is becoming essential to build an efficient and effective engine that can satisfy this attempt, however, my project is to create a light-weight C# engine that can handle two main functional domains which are parsing HTML files and converting a HTML file with embedded C# code fragments to a pure HTML file.

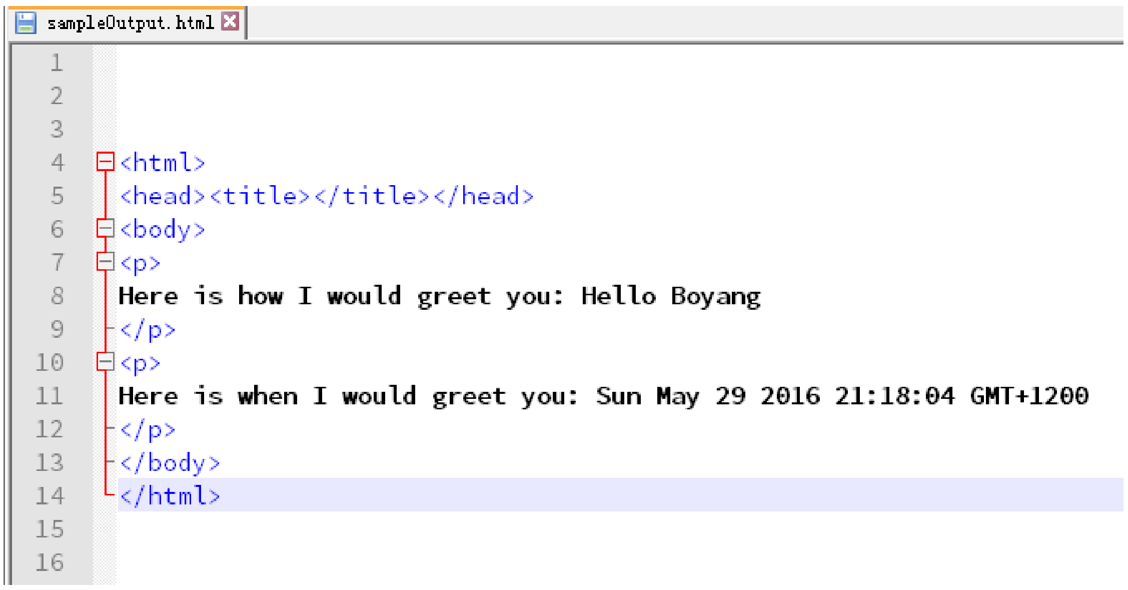
We want to have C# function calls within <? ... ?>, here is a very simple example that shows the expected ability of this C# engine.
E.g.,



```



<html>
<head><title></title></head>
<body>
<p>
Here is how I would greet you: <? GetGreeting("Boyang"); ?>
</p>
<p>
Here is when I would greet you: <? DateTime.Today; ?>
</p>
</body>
</html>


```

Figure 1.1: sample input of the HTML file.

Assuming the call GetGreeting() returned "Boyang", and the call DateTime.Today returned current date and time.



```
<html>
<head><title></title></head>
<body>
<p>
Here is how I would greet you: Hello Boyang
</p>
<p>
Here is when I would greet you: Sun May 29 2016 21:18:04 GMT+1200
</p>
</body>
</html>
```

Figure 1.2: sample output of a HTML file.

This illustrates HTML pages with some embedded C# fragments and these fragments will be replaced by the output of running these fragments.

1.2 Expected Outcomes

--Enhance individual programming skills and formal report writing skills.
--Enhance my collaborative abilities of working one-on-one with supervisor.
--Sharpen my critical and analytical thinking skills.
--Gain individual research skills.

1.3 Report Structure

The remainder of this report will explain some details as follows: the chapter 2 will explain project basics which contains the program language, language Framework and some powerful feature given by that framework. The chapter 3 explains serval aspects of project development. And the last chapter will describe the project work done so far and make a plan for further work in the semester two.

Chapter 2

Project Basics

This section will introduce the basic knowledge and background of my project, it contains the program language, framework and some powerful features or functionality that provided by which framework.

2.1 Programming Knowledge Overview

2.1.1 C#.NET

First of all, C# programming language is a modern language created by Microsoft and it is same as VB.NET, Managed C++, and F# which is a part of .NET languages that capacitate developers to build a diverse range of applications which run on the .NET Framework [1]. Based on acknowledge of C, C++ or Java programming language, it is not hard to recognize similarities of syntax among these languages. However, compared to C++, C# is able to reduce the time that may be taken by users to employs it during development processes due to optimizing the complexities of syntax. In addition, some incredible useful functions which cannot be found in Java while provided by C# such like nullable value types, enumerations, delegates, lambda expressions and direct memory access [2]. Furthermore, some other advantages such as C# supports generic methods and Language-Integrated Query (LINQ) expressions, which means the former convenience facilitates the implementations of specific collection behaviours and the latter one improves the time mobility of developers during writing code.

2.1.2 .NET Framework

The .NET Framework is a software technology created by Microsoft that enables C# programs to run on it. Once you install .NET Framework, it creates a type-safe and object-oriented programming environment which supports developing and running a branch of various applications and XML Web services. The attempt of .NET Framework is not only to provide a capability of orderly accessing code database and Web-based applications but also to afford an interoperability of serval programming languages, which means a consistent code-execution environment is possible to minimise versioning conflicts.

Based on studies of [3], the .NET Framework is combined of two fundamental components which are Common-Language-Runtime (CLR) and Framework-Class-Library (FCL). The CLR is used to compile and run applications, beyond that, it is also used to manage .NET code, memory, exceptions, debugging, code safe verification, and other services. And the FCL is a myriad of predefined classes or reusable types that you can use to define object properties in your programs, these classes provide runtime functionality which can be derived when managing your own code, additionally, other database interactions and features given by FCL make it possible to employs .NET Framework types more efficiently. Third-party libraries or source code produced by programmers also can be merged seamlessly into .NET Framework.

Once the programme source code written in C# is compiled, then a managed assembly or executable file is generated with an extension of .exe or .dll which are stored on the disk. After that, when the C# program is executed, the assembly is load into CLR, and then if the requirements of security features are satisfied completely the CLR would convert it to native machine instructions. The following figure shows the interactions among .NET Framework, assemblies, the CLR, and the FCL in compile-time and run-time of C# programs.

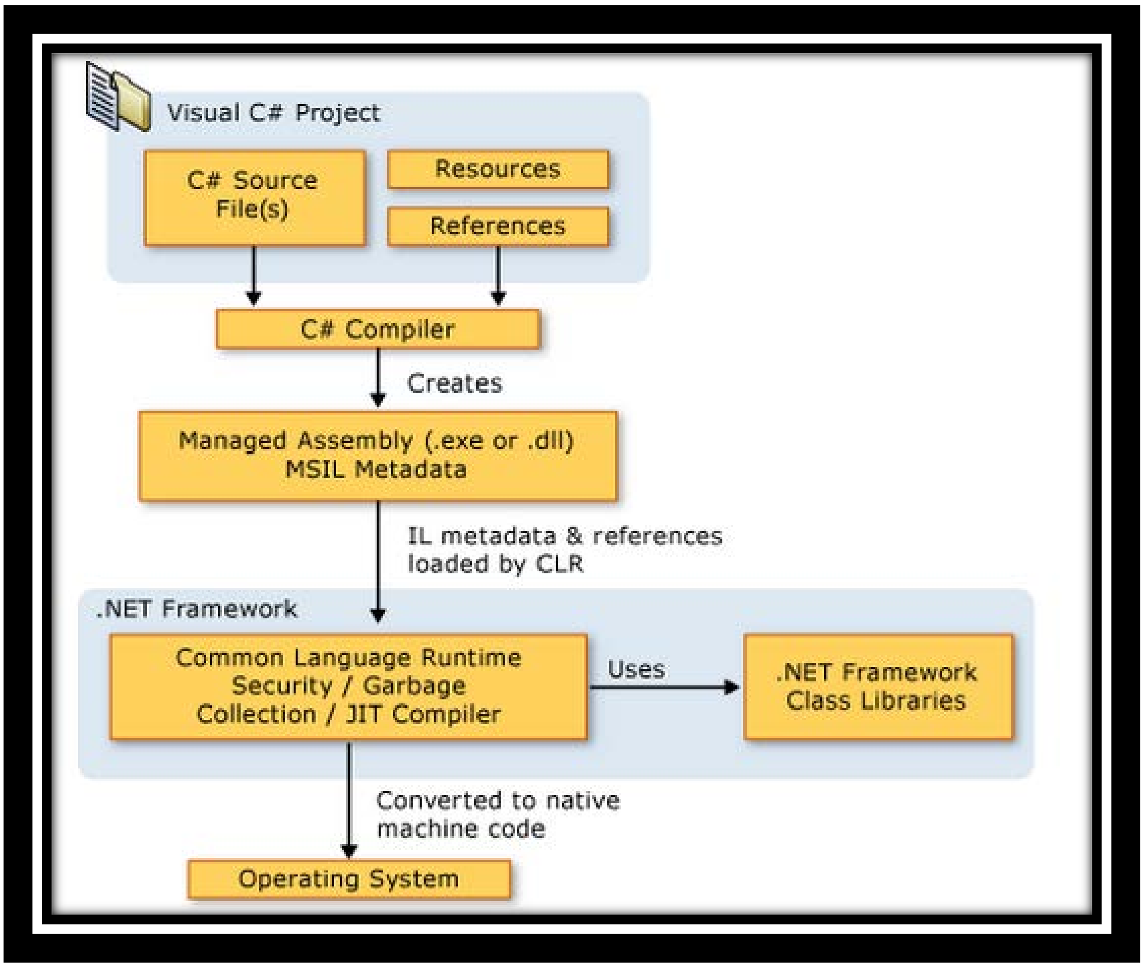

Visual C# Project
C# Source File(s)
Resources
References
C# Compiler
Creates
Managed Assembly (.exe or .dll) MSIL Metadata
IL metadata & references loaded by CLR
.NET Framework
Common Language Runtime Security / Garbage Collection / JIT Compiler
Uses
.NET Framework Class Libraries
Converted to native machine code
Operating System

Figure 2.1: .NET Framework Platform Architectural [2]

There are some further information for deeper understanding .NET Framework which given by [4]. In brief, the .NET Framework is integrated by Common Language Runtime and Framework Class Library, the primary principle concerning a natural phenomenon of runtime can be regarded as code management. To be more specifically, the .NET Framework can not only be hosted by managed components but also unmanaged or third-party runtime hosts, take ASP.NET for instance, it is an example of a managed application whereas the Internet Explorer is an example of an unmanaged application.

The following diagram illustrates the connections between CLR and FCL to the whole system and shows how managed and unmanaged applications implemented in a bigger architectural model.

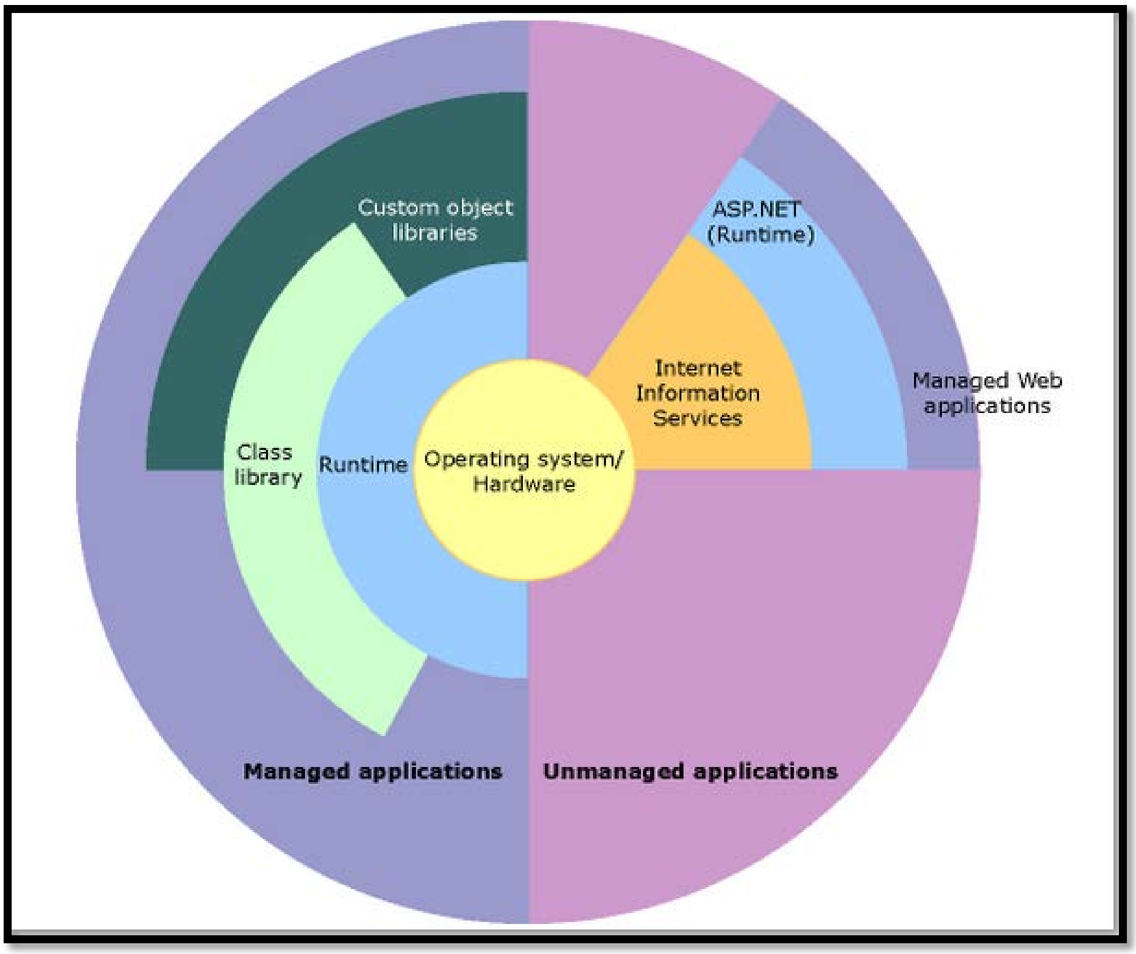


Figure 2.2: Overview of .NET Framework [4].

2.1.3 Differences between Framework and Library

The last thing I would like to finish on is the difference and relation between library and framework. Actually, the library is a collection of predefined classes which have been written by other programmers, a library gives a lot pieces of functionality that you may pick up and choose from. Whereas, typically framework is more complicate than library, it introduces an architecture that your application will follow. The design decisions and code structure are based on what framework you choose, which means once a framework is chose then your code will be called by the framework appropriately. In fact, it is more likely to regard the framework as a top level which is centred by many libraries. Basically, “Inversion of Control” is the core difference between a framework and a library, the following diagram shows control in different ways.

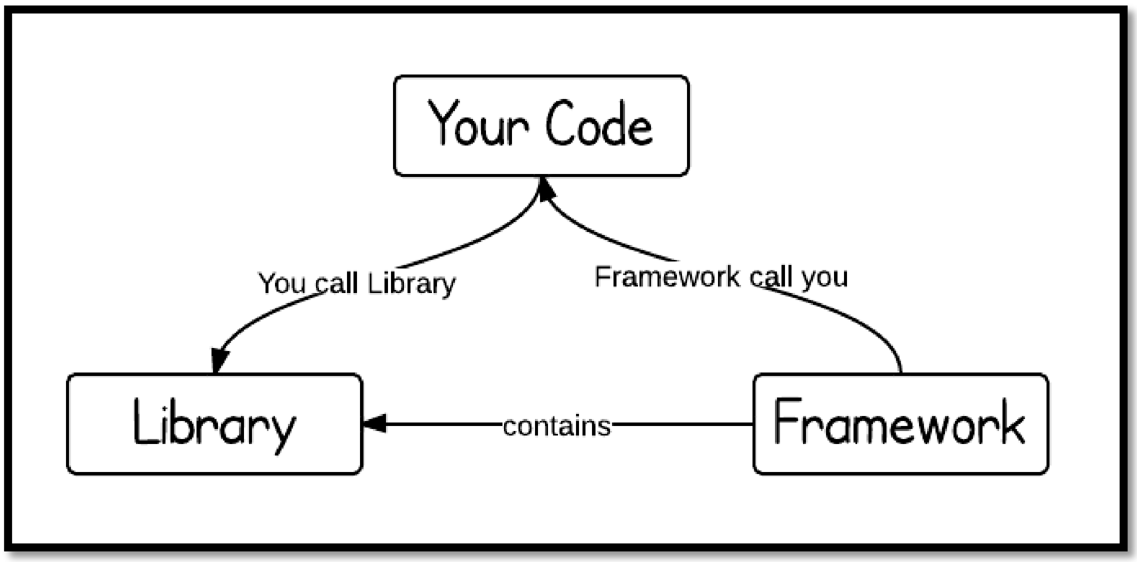


Figure 2.3: Library vs. Framework [5].

2.2 Reelection and Dynamically Loading

2.2.1 Reflection in .NET

Reflection is a functionality that enables computer program to fetch type (assembly) at runtime, which means it is able to estimate and modify the structure and behaviours of the program that cannot be achieved at compile time. There is a list of useful classes given by *System.Refelction* namespace that allows you to collect information within assemblies, the information could be types, properties, methods and so on. Additionally, all types such like types of classes, types of interfaces, and value types can be acquired from loaded assemblies by using *System.Type.* .NET Reflection makes it possible to dynamically create an instance of a type, bind the type to an existing object, or get a type of existing object. In the next you can invoke the type's methods or access its fields and properties [6].

Currently, in my project, the embedded functional fragments extracted from HTML file will be merged into a single C# file, which is named "MyOwnMethods.cs" or any other proper names, my project will be presented as a Command-Line program, here is a Command:

```
CAR.exe -cs MyOwnMethods.cs SomeonesLib.dll -output SampleOutputHTML.html
SampleInputHTML.html
```

When running the program, the code in MyOwnMethods.cs will be loaded as assembly, the program CAR will compile the code therein, and then run it. In this case, in order to replace the embedded code fragments by the output of running these fragments, .NET Reflection allows you to invoke the type of existing object in this C# file as well as invoking the type's methods at runtime.

As we know, in .NET Reflection the combination of *System.Refelction* namespace and *System.Type* class allows you to reflect over many other aspects of a type. Before using Reflection, it is important to know what these two are made of, the following figure shows a road map of .NET Reflection.

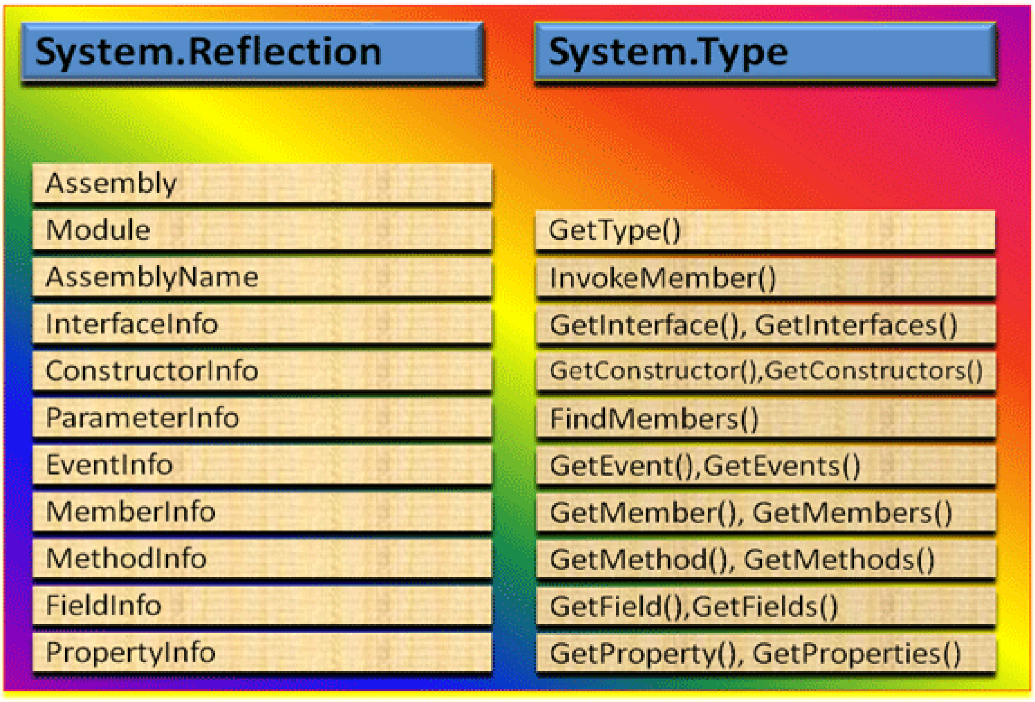
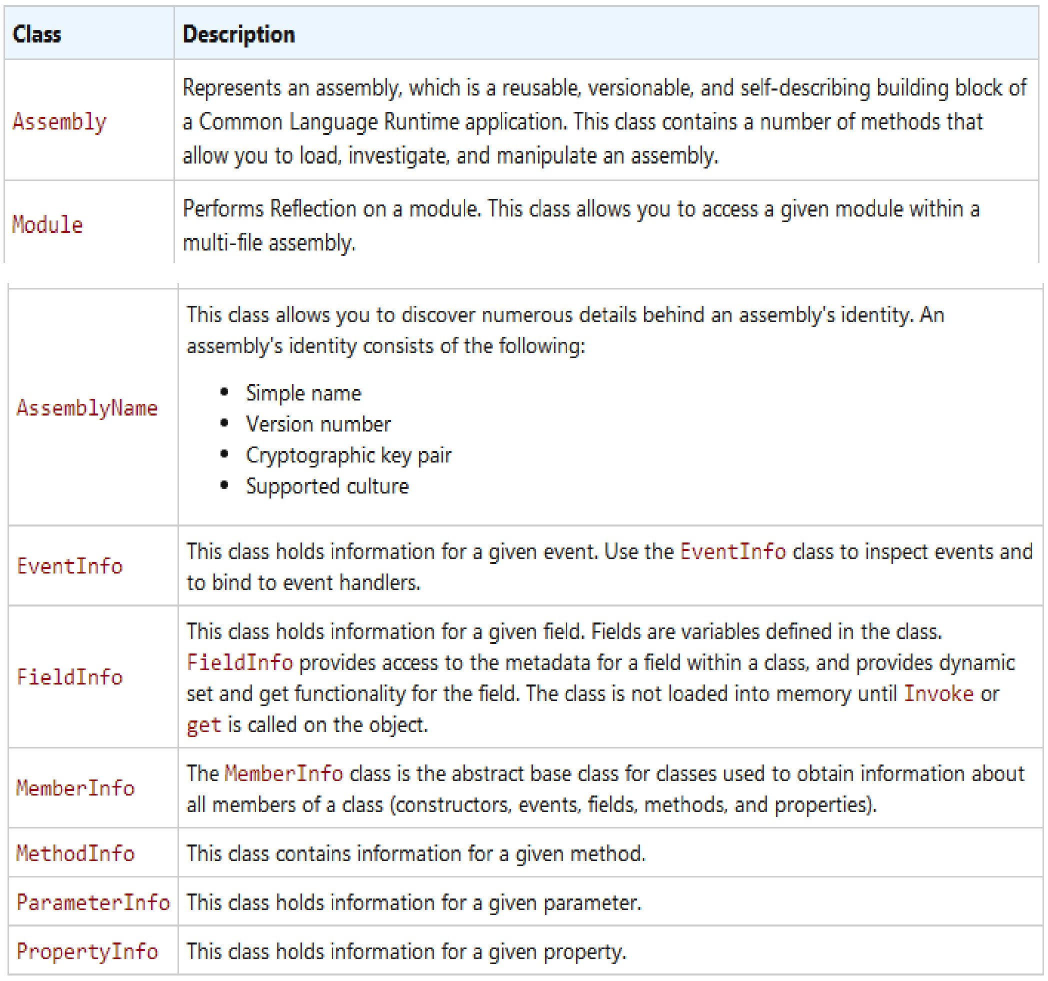


Figure 2.4: .NET Reflection Road Map [7].

According to above road map, there are some commonly used classes such as Assembly, Module, InterfaceInfo, ParameterInfo, and MethodInfo and so on. The details about abilities of them are shown below:



Class	Description
Assembly	Represents an assembly, which is a reusable, versionable, and self-describing building block of a Common Language Runtime application. This class contains a number of methods that allow you to load, investigate, and manipulate an assembly.
Module	Performs Reflection on a module. This class allows you to access a given module within a multi-file assembly.
AssemblyName	This class allows you to discover numerous details behind an assembly's identity. An assembly's identity consists of the following: • Simple name • Version number • Cryptographic key pair • Supported culture
EventInfo	This class holds information for a given event. Use the EventInfo class to inspect events and to bind to event handlers.
FieldInfo	This class holds information for a given field. Fields are variables defined in the class. FieldInfo provides access to the metadata for a field within a class, and provides dynamic set and get functionality for the field. The class is not loaded into memory until Invoke or get is called on the object.
MemberInfo	The MemberInfo class is the abstract base class for classes used to obtain information about all members of a class (constructors, events, fields, methods, and properties).
MethodInfo	This class contains information for a given method.
ParameterInfo	This class holds information for a given parameter.
PropertyInfo	This class holds information for a given property.

Figure 2.5: Class Description.

In the other hand, the Type class represents a type in the **Common Type System** (CLS), and it is more capable of accessing metadata as well as representing type declarations. The type information can be obtained from type declarations such as class types, value types, interface types, and enumeration types through three ways. First of them is ***System.Object.GetType()***, this method can only be achieved when you have compile time knowledge of the type, and it will return a Type object that on behalf of the type of an instance.

The second approach is ***System.Type.GetType()***, this is more flexible than former one, which means this way enables you to get a type with particular parameters and the last one is C# ***typeof*** operator. The Figure 2.6 is the outline of these methods.

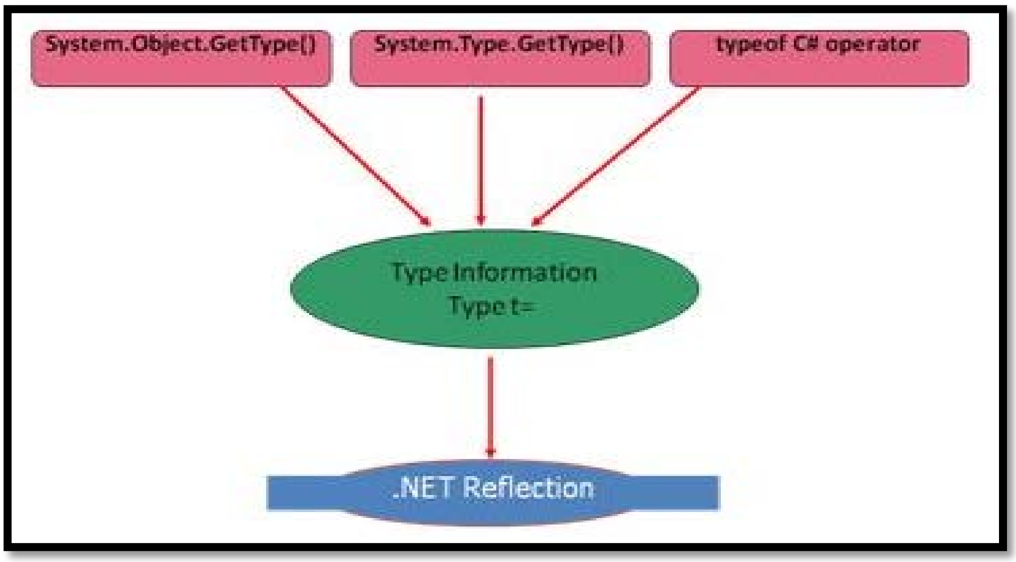


Figure 2.6: Obtaining Type Information [7].

2.2.2 Dynamically Loading Assembly

In .NET Framework, dynamic assemblies loading can be regarded as an advanced topic of Reflection. Dynamically loading libraries is widely used in programming, this functionality is presented by using Dynamic Link Library (DLL) in native C#. This capability makes the applications become more modifiable, which means it is able to add some specified features to existing computer program or modify them at runtime without need to re-compile them again. Likewise, dynamic assembly loading is another powerful tool provided in .NET Framework.

Dynamically loading can only occur when there is a medium for communication between applications and assembly components. This can be realized with the use of a commonly agreed interface, the content of interface can be changed not until entire lifecycle is over, since any change of the interface will cause whole application and all components to be completely recompiled [7].

In detail, based on .NET Framework, once the application has been created, then we can dynamically load assemblies, where these assemblies could contain more one class which implement interface, however, the application has no visibility of the classes implemented in assemblies, thus, the interface between an application and assemblies builds a bridge for them. The Figure 2.8 shows the relationship.

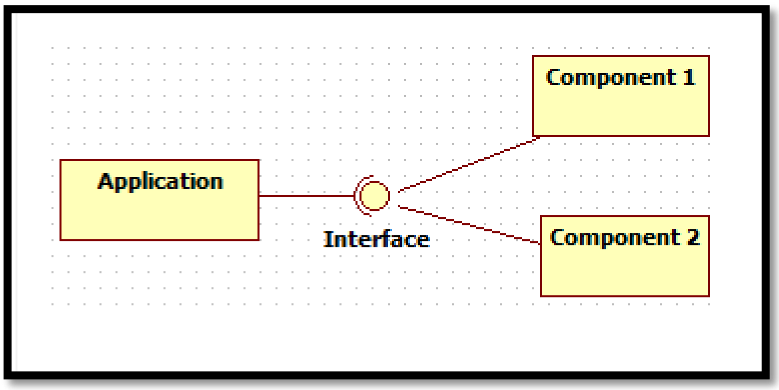


Figure 2.8: Dynamically Loading.

Chapter 3

HTML1st Design

This section will discuss several core parts of HTML1st development.

3.1 Scripting Language

3.1.1 Scripting Language Overview

Scripting languages or scripts are sometimes considered as high-level languages. By comparing scripting and normal programming, scripts are interpreted while programs are compiled, which means they are executed in different ways. To be more specifically, the scripts are interpreted by another program at runtime, where these scripts are distinct from the core language used in that application, they are probably written in different languages [10].

Scripting languages make it possible to integrate and communicate with other programming languages, take dynamic Web pages for instance, JavaScript is one of mostly used client-side scripting languages, which is usually embedded within HTML, it enables you add extensive capabilities to Web pages, in this case, Web pages become more flexible. Additionally, there are some other commonly used scripting languages such as PHP, ASP, JSP, Perl, Python, and Ruby and so on [8].

In recent years, there is a trend to develop an application by using the conjunction of scripting languages and system programming languages, each of them can be looked as a complement of another one. Scripting language is more likely to be designed for gluing [9], the attempt of glue language is to connecting the software components with scripts. Another very powerful feature of the scripting languages is typeless, which means an instance and hold a variable type at one moment and another the next.

In my project, there is a step that using C# program to compile the C# code fragments extracted from HTML pages, where those fragments are scripts. Typeless enhances the ability of scripting languages that allows data and code to be interchanged, so that the HTML1st engine can execute the fragments as another program on the fly.

In the end, it is essential to talk about a set of trade-offs that exist in scripting languages. With the increasingly wide use of scripting languages in such fields like: graphical user interfaces (GUI), Internet, and framework components. For example, a growth of the Internet interactions have been realized by scripting languages, as the scripting technology makes it easier to exchange things between database and web browser. It is really efficient and attractive when application is large and complex [9]. Moreover, scripting languages are easy to learn and use, it allows you to manipulate dynamic activities and modify stuff that are already done. An interactive web page can be created by user with less effort, which improves the speed of communicative response. Furthermore, scripting languages strengthen the productivity of developers and reuse of components. In contrast, it is more time-consuming due to scripts are interpreted at runtime and bot compiled into machine code, meanwhile, the security concerns cause the inconsistent distribution.

3.1.2 Client-Side Scripting

Traditionally, the Web browser is a client-side environment that allows scripts to run on the end users' computers. Suppose there are some dynamic or custom web contents need to be presented on someone's computer, but a beautiful Web page only consists of HTML without any scripts, the page cannot do anything but just sit there. Scripts facilitate the ability that web pages can have varying and changing content depending on user inputs. The best solution for interaction between end users and web pages is client-side scripting. Sometimes, documents produced after running server scripts, which contains some client-side scripts, then they are delivered to the user's computer over internet and run directly in the browser, the only requirement at client-side is the browser understands what theses scripts mean [12].

Usually, client-side scripts are embedded in a HTML or XHTML document, which is designed to create instructions for the browser to follow in response to user actions such as window or menu display, button or mouse events or keyboard typing. As we know, the most popular client-side scripting language is JavaScript, this language is a member of object-oriented languages. Within web browser, the aim of JavaScript is to manipulate the elements on a web page and control behaviours such like the occurrence of an event. Although, the JavaScript is easy to learn and understand, but before we start using it, we need to be aware of its connections with HTML. The elements in the HTML document are constructed in Document Object Model (DOM), this structure is presented as s hierarchical architecture style. And this structure is applied to organize the objects of a web content (see Figure 3.1). However, different DOMs give different flexibility levels to design a web page when you implement JavaScript.

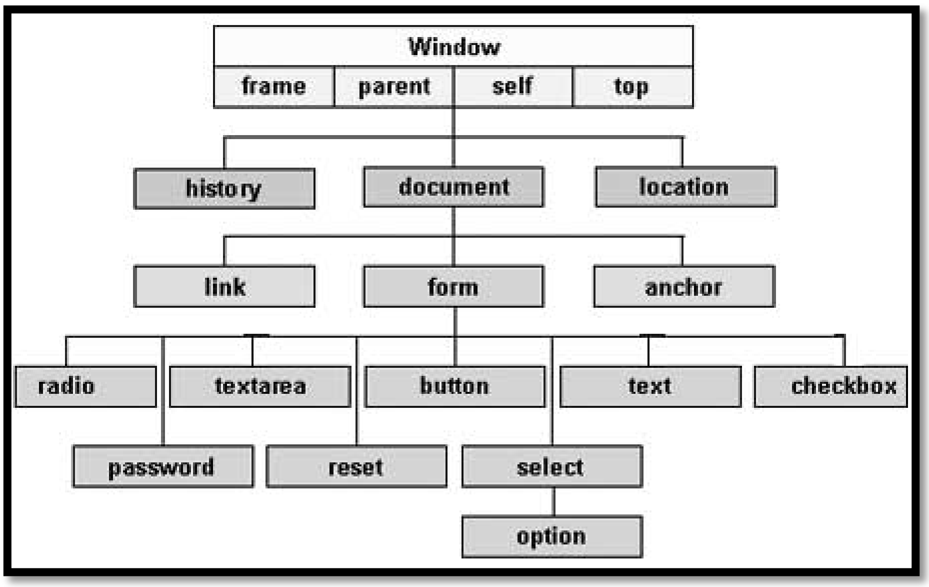


Figure 3.1: Simple hierarchy of DOM [11].

Compared to server-side scripting, the client-side scripting works in the front of a website, user can see whatever the Client-side code have done and the stuffs have been presented out. Once the document has been transferred from back-end, if there is no additional requests in response to Web server, the specified contents of code will be processed by browser in an isolated circumstance until a new request was sent back to the server. In this case, the immediately interactive communication between the Web browser and end users has speeded up sharply. In the front-end web development, the Client-side scripting languages is a mix use of HTML, JavaScript, and cascading style sheet (CSS), where the CSS is a file that applied to style the way the page looks. Additionally, there are varying kinds of JavaScript Frameworks such like AngularJS, JQuery, Node.js, and AJAX.

3.1.3 Server-Side Scripting

In contrast to Client-side scripting, the Web server provides an environment that allows a Server-side scripts to run whenever a user’s request is received. Usually, there are database or data stores on the server side, the primary advantage to Server-side scripting is granting permission for accessing the database when specified information required by users. In other words, a dynamic web page can be generated by running scripts on Server-side based on custom requirements. The whole web development is a client-server system, any web browser resides on a computer can be regarded as a client and the web pages which have been requested will be sent back to the client [15]. This process can be shown as following diagram:

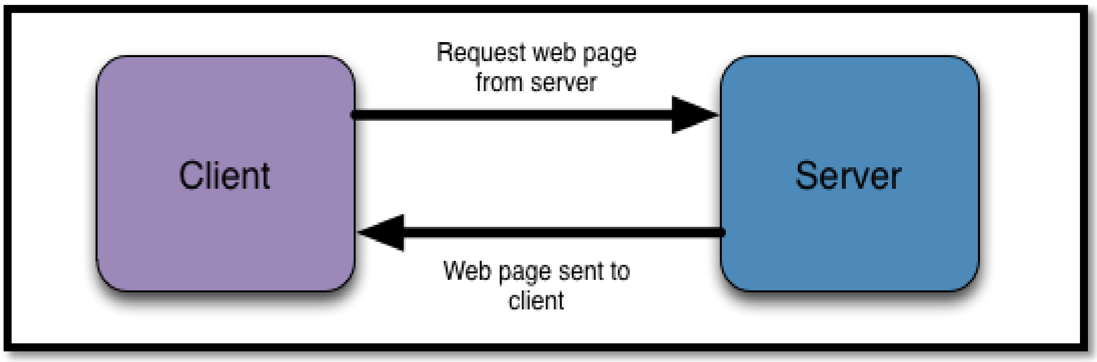


Figure 3.2: Client-Server System [14].

This diagram shows that client can only require static web pages from the server, but these days most websites on the Internet have dynamic contents, the common gateway interface (CGI) provides a functionality that enables the web server to run the scripts and automatically process a set of instructions. Typically, a dynamic page would have an extension such as **.cgi** or **.php**.

Once a request with one of these extensions, it will be delivered from the web server to CGI, and then the scripts will be correctly interpreted and executed. At last, the standard HTML page will be sent back to the client, the end user's browser only needs to worry about what results are presented to users rather than the underlying script used to generate this web page [14]. The following diagram shows the extensive ability of web server:

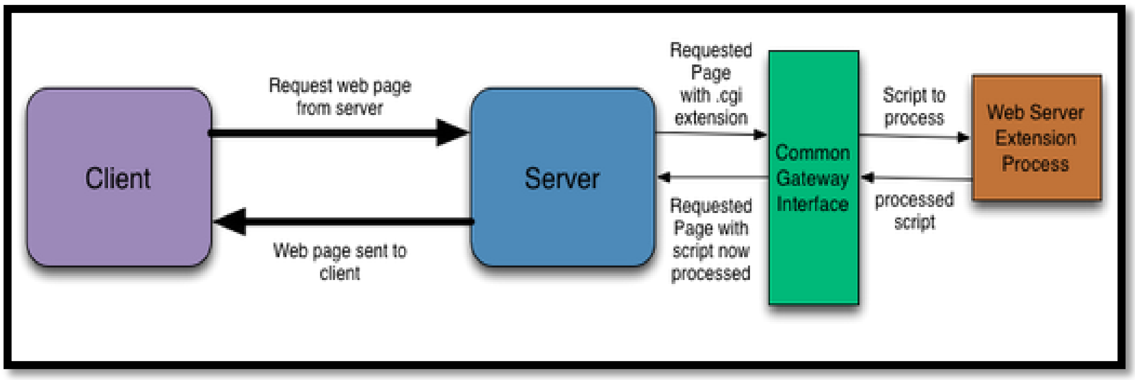

Client
Request web page from server
Web page sent to client
Server
Requested Page with .cgi extension
Requested Page with script now processed
Common Gateway Interface
Script to process
processed script
Web Server Extension Process

Figure 3.3: CGI with Web Server [14].

In the development of back-end, there consists three core parts: server, database, and APIs, where the APIs structure how data is exchanged between a database and any software accessing it. The server can be any remote powerful computer located at anywhere. And there is a back-end software written by back-end web developers via server-side scripting languages. Then, a fast and secure channel has been created for exchanging information among user, server, and database. Server- side scripts process requests and pull what they need from the database, then update information for the end users. For instance, if user want to see his (or her) online banking details, after login step, the request is sent to server, the server-side scripts will interact with the database to collect the specified account information the user needs, then process it on the server, at last, the dynamic page will be updated and sent back to browser.

In my project, the C# engine could be looked as the software on server side to build your website behind screen, using it to parse HTML pages and figuring out embedded C# code fragments within it, we should treat them as scripting language, it is more likely to using the C# engine to compile C# scripts and then execute them. Eventually, a pure HTML page will be collected and render it into a human viewable website.

There are some popular server-side languages such as PHP, C#, Ruby, C++, Java, and Python, and their Frameworks such like Ruby on Rails, ASP.NET, Django, and Node.js: JavaScript. In conclusion, the Web development is combined of front-end and back-end development, any website should base on three components: the server, the client, and the database, the following two diagrams illustrate an overview of client-side and server-side working flow:

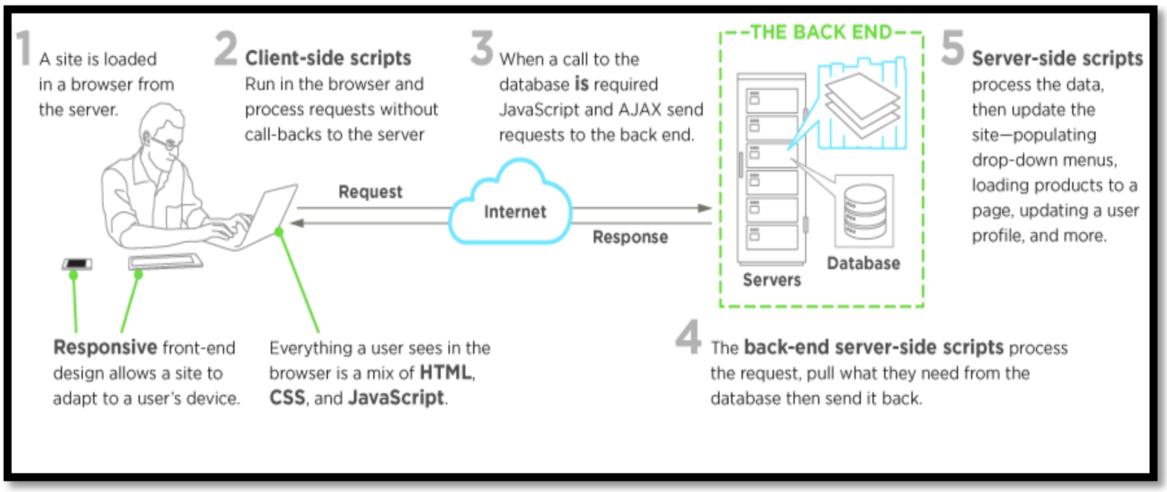

1 A site is loaded in a browser from the server.
2 Client-side scripts Run in the browser and process requests without call-backs to the server
3 When a call to the database is required JavaScript and AJAX send requests to the back end.
THE BACK END
5 Server-side scripts process the data, then update the site—populating drop-down menus, loading products to a page, updating a user profile, and more.
Request
Internet
Response
Servers
Database
Responsive front-end design allows a site to adapt to a user's device.
Everything a user sees in the browser is a mix of HTML, CSS, and JavaScript.
4 The back-end server-side scripts process the request, pull what they need from the database then send it back.

Figure 3.4:Front-End Development [13].

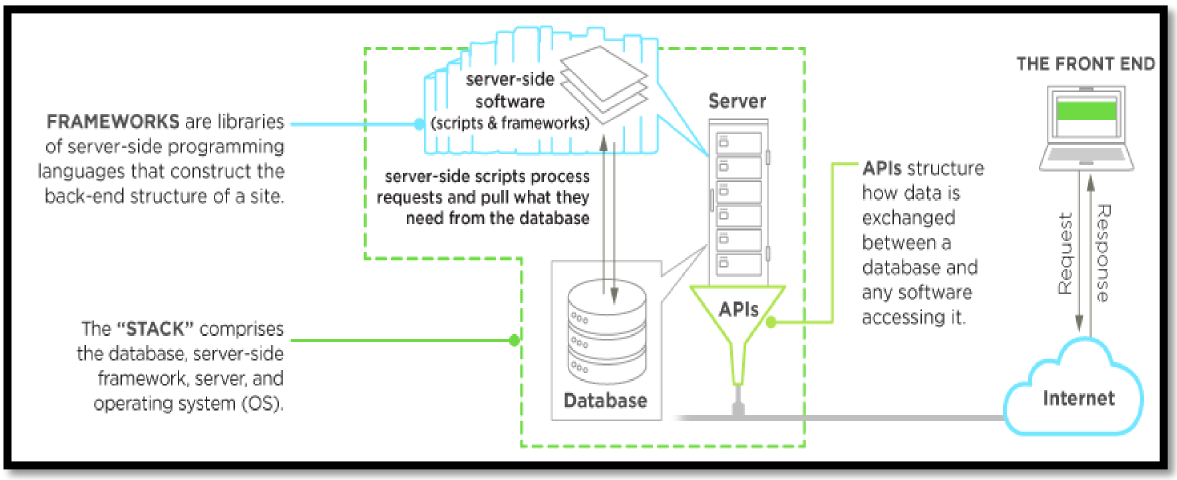

server-side software (scripts & frameworks)
Server
THE FRONT END
FRAMEWORKS are libraries of server-side programming languages that construct the back-end structure of a site.
server-side scripts process requests and pull what they need from the database
APIs structure how data is exchanged between a database and any software accessing it.
Request
Response
APIs
The "STACK" comprises the database, server-side framework, server, and operating system (OS).
Database
Internet

Figure 3.5: Back-End Development [16].

3.2 Treating C# like A Scripting Language

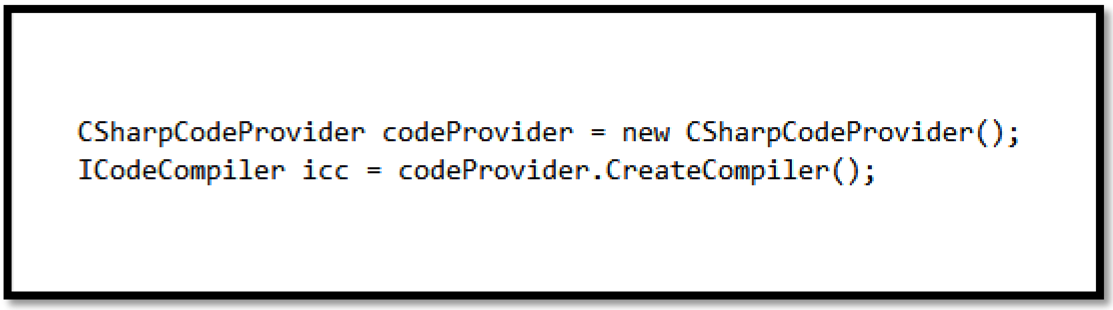
In my project, C# code fragments embedded in HTML page are very simple, most of them are exact some functions or methods with a parameter, the problem here is how we can process them. Suppose there has a single file that contains all the functions which are embedded in HTML file, I wonder if it is possible to look them as a scripting language, then they can be interpreted at runtime. As a consequence, we can dynamically manipulate and modify the code during runtime.

What I found was that C# does indeed have the capability to accomplish this task, it allows you to load C# from anther file and execute them therein. However, it is a little bit more complicated in this case due to the C# is a complied language, the code needs to be compiled into an assembly before using it. Once the C# code fragments are loaded as an assembly, then we can invoke a type of existing object and invoke the type's methods at runtime by Reflection in .NET Framework.

It is likely to use C# to compile C#, this powerful feature is provided by .NET Framework in Microsoft.CSharp and System.CodeDom.Compiler namespaces without any third-party libraries [17].

For compiling the C# code on the fly at runtime, there are several core steps you need to follow:

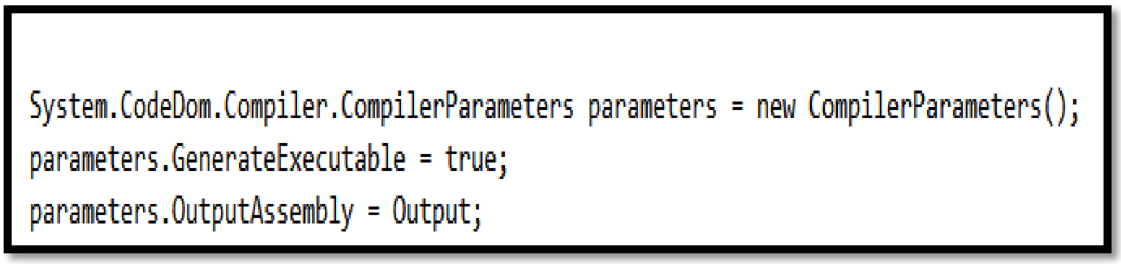
1. To programmatically compile your code you need to create a C# compiler, the Figure 3.6 shows the compiler created by using an instance of **CSharpCodeProvider** class:



```
CSharpCodeProvider codeProvider = new CSharpCodeProvider();
ICodeCompiler icc = codeProvider.CreateCompiler();
```

Figure 3.6: C# compiler.

2. Then you can create parameters of the compiler by using an instance of **CompilerParameters** class, which contains a set of parameters that will be passed to compiler when compiling your code, additionally, you can also define whether your code will be generated only in the memory or into the DLL or EXE file (see Figure 3.7).



```
System.CodeDom.Compiler.CompilerParameters parameters = new CompilerParameters();
parameters.GenerateExecutable = true;
parameters.OutputAssembly = Output;
```

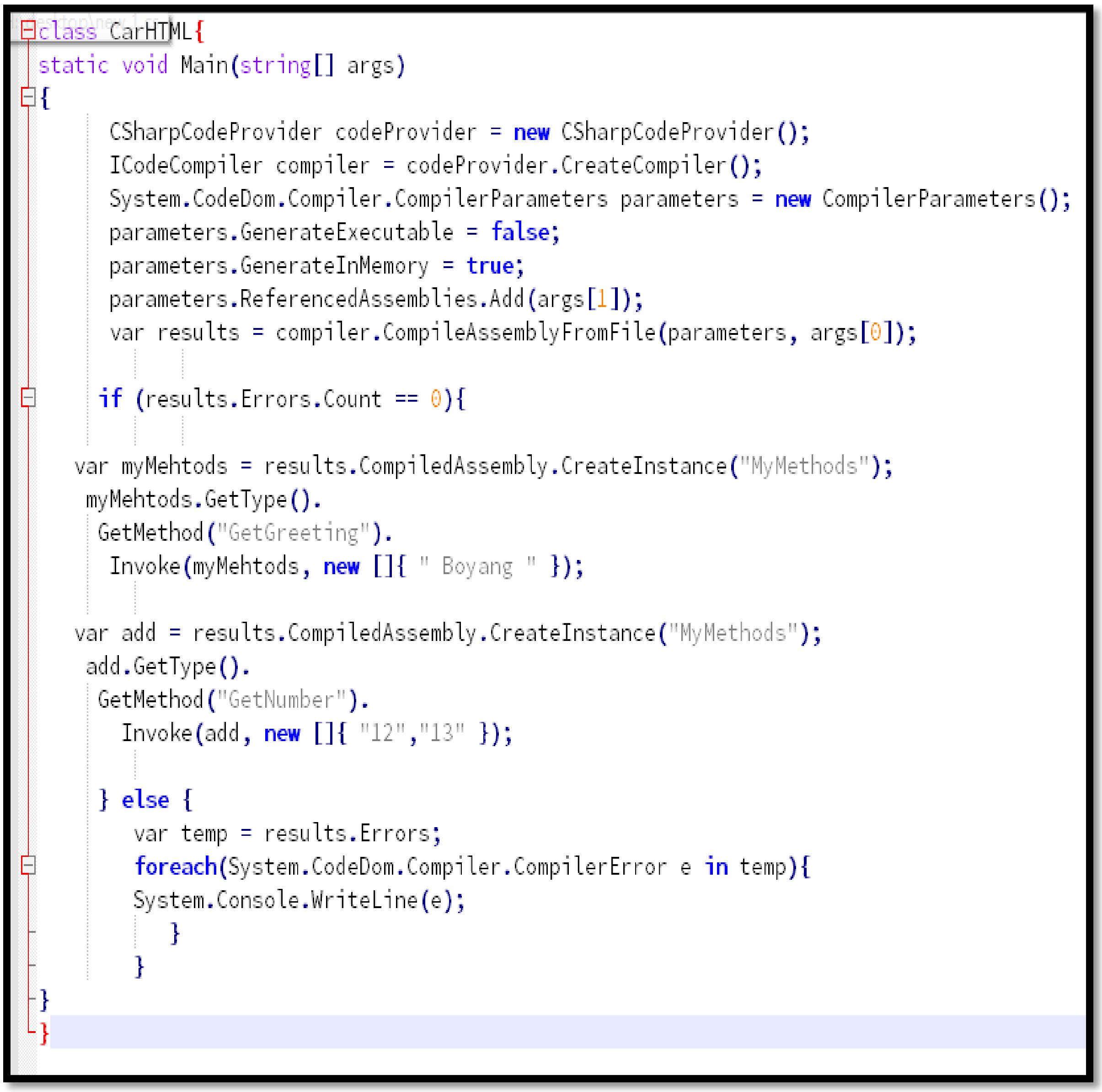
Figure 3.7: Parameters.

3. Define the parameters, you can add any library to your compiler by using **parameters.ReferenceAssemblies.Add()**;

4. Compile your assembly: **CompilerResults results =provider.CompileAssemblyFromSource(parameters,SourceString);**

5. Check errors and use the compiled code, once your code has been compiled into an assembly, you can use that assembly to create instances of classes from your source code and use reflection to invoke methods and get or set properties of those classes.

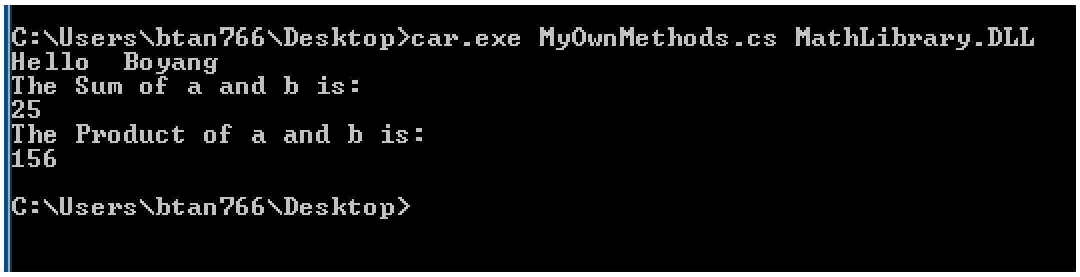
Here is a small demo program of this part:



```
class CarHTML{
static void Main(string[] args)
{
        CSharpCodeProvider codeProvider = new CSharpCodeProvider();
        ICodeCompiler compiler = codeProvider.CreateCompiler();
        System.CodeDom.Compiler.CompilerParameters parameters = new CompilerParameters();
        parameters.GenerateExecutable = false;
        parameters.GenerateInMemory = true;
        parameters.ReferencedAssemblies.Add(args[1]);
        var results = compiler.CompileAssemblyFromFile(parameters, args[0]);

       if (results.Errors.Count == 0){

     var myMehtods = results.CompiledAssembly.CreateInstance("MyMethods");
      myMehtods.GetType().
       GetMethod("GetGreeting").
        Invoke(myMehtods, new []{ " Boyang " });

     var add = results.CompiledAssembly.CreateInstance("MyMethods");
      add.GetType().
       GetMethod("GetNumber").
         Invoke(add, new []{ "12","13" });

       } else {
          var temp = results.Errors;
          foreach(System.CodeDom.Compiler.CompilerError e in temp){
          System.Console.WriteLine(e);
             }
          }
}
}
```

Figure 3.8: using C# compile C#.

Output:



```
C:\Users\btan766\Desktop>car.exe MyOwnMethods.cs MathLibrary.DLL
Hello  Boyang
The Sum of a and b is:
25
The Product of a and b is:
156

C:\Users\btan766\Desktop>
```

3.3 HTML Parser

3.3.1 HTML Agility Pack

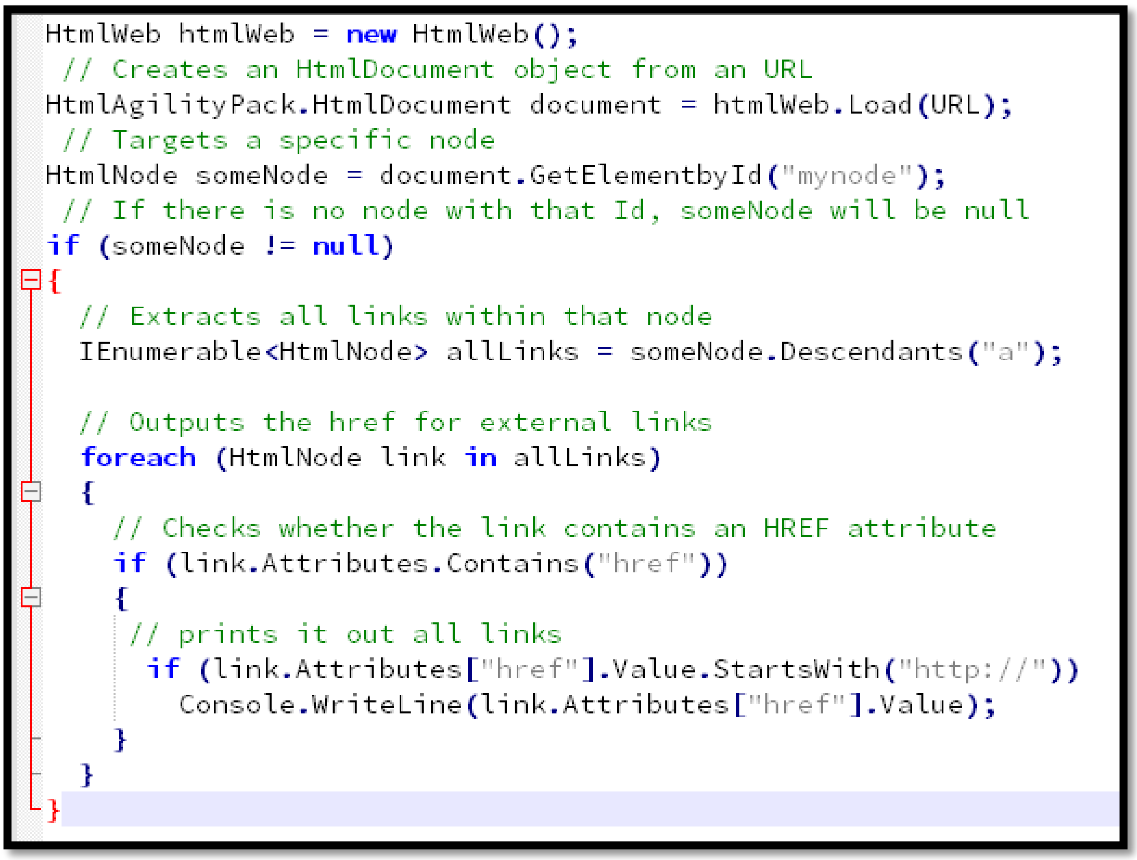
The aim of HTML parsing is to extract some useful and powerful information from an HTML document for further useable fields. However, the HTML is an irregular language, it is crucial to find a way that easily read and modify the HTML string code, which means it is able to identify the format and syntax of a string of symbols, in other words, it is a syntactic analysis of HTML documents following rules or a formal grammar. HTML Parsers is such a tool to accomplish this kind of tasks, and they can be presented as a computer program or a library given by a Framework.

A HTML parser can be written in any popular language, but in my project, it is to find an appropriate way to parse HTML using C#. What I found was the most widely used and efficient way named HtmlAgilityPack, this pack is a .NET code library that allows you to parse a real-world HTML and it is very tolerant in handling elements, text, attributes, and other markups within HTML even the format is invalid.

In fact, HTML is a structured document format with a varying kinds of very clearly defined rules. Basically, you can create a C# application to parse a HTML page with regular expressions, but it seems more efficient when you use a DOM-based approach with a functionality such as LINQ (or XPath) [18]. .NET Framework provides an HtmlDocument class, along with HtmlElement, which allows you to access data by calling DOM methods such like GetElementById and GetElementByTagName. However, there is no such a constructor when you build an instance of HtmlDocument, even if you can use XmlDocument and XmlNode to read from or write to XHTML documents, it also needs a third-party library to check the validity of format and the correctness of markup [19].

HTML Agility Pack is a free and open source library whose attempt is to load the HTML from either a file or a remote website, and then parse it. The HtmlDocument and HtmlNode classes are provided by HtmlAgilityPack, the capabilities of these classes are quite similar to that of XmlDocument and XmlNode classes. One attractive feature of HtmlAgilityPack is that it constructs a Document Object Model (DOM) view of the HTML document being parsed, which makes programmers easier to read through the documents and move from parent nodes to their child nodes. Secondly, there is no need to check markup validity due to HtmlAgilityPack will take care of making everything valid by closing unclosed tags and fixing other markup errors. Moreover, the HtmlAgilityPack allows you to return or retrieve specified nodes through the use of XPath expressions [19].

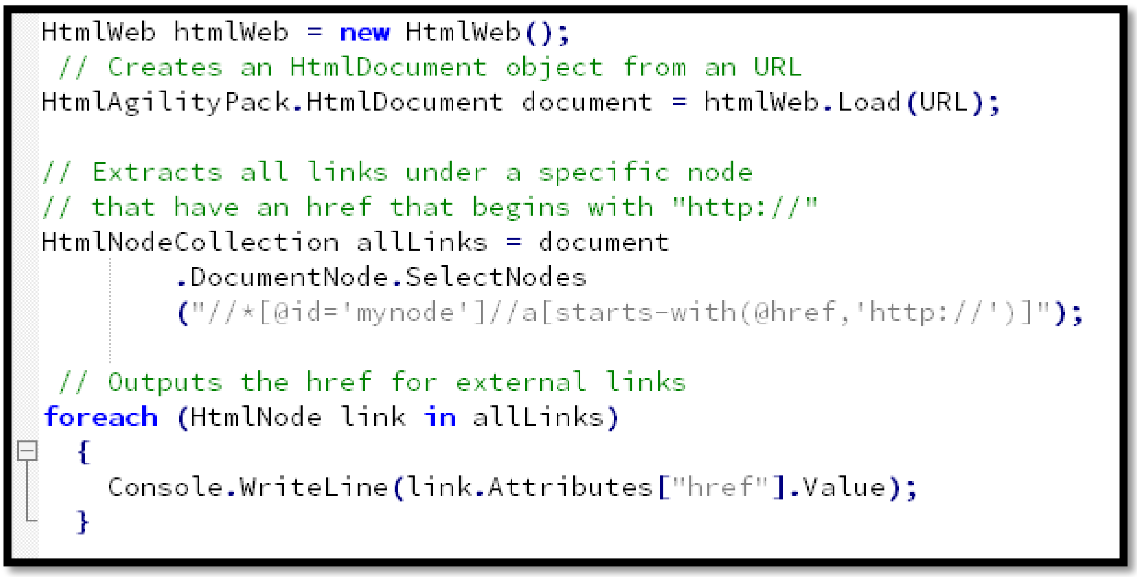
The figure 3.9 shows example for parsing with the use of HtmlAgilityPack:



```
HtmlWeb htmlWeb = new HtmlWeb();
 // Creates an HtmlDocument object from an URL
HtmlAgilityPack.HtmlDocument document = htmlWeb.Load(URL);
 // Targets a specific node
HtmlNode someNode = document.GetElementbyId("mynode");
 // If there is no node with that Id, someNode will be null
if (someNode != null)
{
   // Extracts all links within that node
   IEnumerable<HtmlNode> allLinks = someNode.Descendants("a");

   // Outputs the href for external links
   foreach (HtmlNode link in allLinks)
   {
     // Checks whether the link contains an HREF attribute
     if (link.Attributes.Contains("href"))
     {
      // prints it out all links
       if (link.Attributes["href"].Value.StartsWith("http://"))
         Console.WriteLine(link.Attributes["href"].Value);
     }
   }
}
```

Figure 3.9: Parsing with HtmlAgilityPack.

A more efficient way with the use of XPath expressions:



```
HtmlWeb htmlWeb = new HtmlWeb();
 // Creates an HtmlDocument object from an URL
HtmlAgilityPack.HtmlDocument document = htmlWeb.Load(URL);

// Extracts all links under a specific node
// that have an href that begins with "http://"
HtmlNodeCollection allLinks = document
        .DocumentNode.SelectNodes
        ("//*[@id='mynode']//a[starts-with(@href,'http://')]");

 // Outputs the href for external links
foreach (HtmlNode link in allLinks)
  {
    Console.WriteLine(link.Attributes["href"].Value);
  }
```

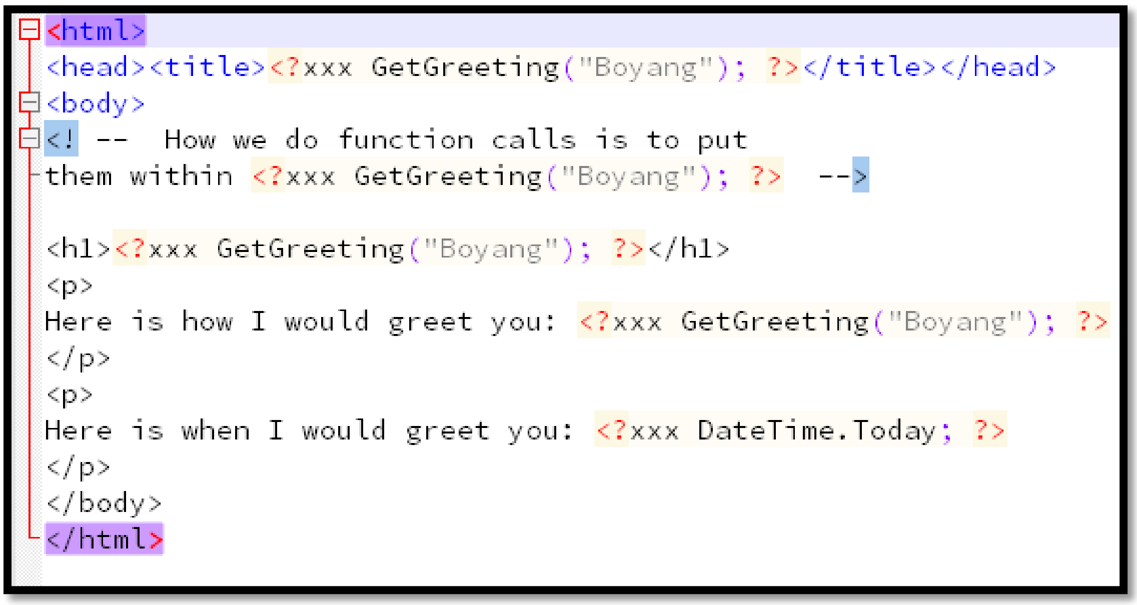
Figure 3.10: XPath expressions.

3.3.2 Processing Instruction

Processing Instructions are special tags with instructions to software applications, which are in an SGML and XML node type. An XML processing instruction is enclosed within <? and ?>, it is typically composed of a target and some string value. The most common use of a processing instruction is to represent a XML style sheet at the beginning of an XML document: <? xml-stylesheet type=”text/xsl” href=”style.xsl” ?> . Sometimes, the same syntax has been used in a XML declaration: <? Xml version=”1.0” encoding=”UTF-8” ?>, but this is not a processing instruction [20].

It was great to find out code fragments embedded in HTML file could not be simply looked as elements or attributes of a node, since the functions take place within symbols “<?” and “?>”, for instance, <? Function(); ?>, we should regard them as Processing Instructions (PIs). Then, we need to check whether the HTML Agility Pack can identify Processing instructions or not. Meanwhile, check to see if HTML Agility Pack can ignore the set of pseudo-strings using same syntax during parsing documents.

Firstly, I have modified the sample input of an HTML file, following figure shows a simple html file embedded with some PIs:



```
<html>
<head><title><?xxx GetGreeting("Boyang"); ?></title></head>
<body>
<! --  How we do function calls is to put
them within <?xxx GetGreeting("Boyang"); ?>  -->

<h1><?xxx GetGreeting("Boyang"); ?></h1>
<p>
Here is how I would greet you: <?xxx GetGreeting("Boyang"); ?>
</p>
<p>
Here is when I would greet you: <?xxx DateTime.Today; ?>
</p>
</body>
</html>
```

Figure 3.10: PIs.

Where the “xxx” is a target of the Processing Instruction, it can be named according to your purpose of each function. Unfortunately, HtmlAgilityPack has no such an ability to identify Processing Instructions which are embedded in the HTML documents. So under this situation, then we can probably treat HTML as XML and use an XML parser to Identify Processing Instructions. As the Processing Instructions are exposed in the Document Object Model (DOM), thus, you can use an XPath expression to get PIs with the ‘processing-instruction ()’ command. After a long process of trial, I found that a well formatted HTML file cannot be parsed by a XML Parser without exceptions, as the HTML and XML are not possible to be regarded as same thing under this condition.

Chapter 4

Conclusion

This section will give a conclusion to current studies of my project and also conclude a plan for further work in the next semester.
As so far, based on the knowledge of Reflection in .NET Framework and awareness of Scripting languages, I have successfully built a small C# program, and use it to load an assembly that contains some C# code fragments, and then execute them as part of my program, it does invoke a type's method at runtime. Additionally, I made another assembly which is just a dynamic link library that contains some powerful functionalities, and link it to my program at runtime. Moreover, I have found a good C# parser called HTML Agility Pack, which is code library given by .NET, but it has a limitation on identifying Processing Instructions (PIs), currently, I still have not found a solution to make a breakthrough. The key problem needs to be addressed in the next semester is to fix the above issues that I have talked about, I suppose there may have two ways: create an own HTML parser based on current ones or introduce a new tag like <source> into HTML instead of enclosing the C# fragments within <? ?>.

Bibliography

[1] C# and .NET Programming https://msdn.microsoft.com/en-us/library/orm-9780596521066-01-01.aspx#

[2] Introduction to the C# Language and the .NET Framework https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

[3] .NET https://www.microsoft.com/net

[4] Overview of the .NET Framework https://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx

[5] Library vs. Framework? http://www.programcreek.com/2011/09/what-is-the-differencebetween-a-java-library-and-a-framework/

[6] Introduction to Reflection API https://dotnetcademy.net/Learn/4/Pages/1

[7]Reflection in .NET

http://www.csharpcorner.com/uploadfile/keesari_anjaiah/reflection-in-net/

[8] Scripting language http://searchwindevelopment.techtarget.com/definition/scripting-language

[9] J. K. Ousterhout, "Scripting: Higher level programming for the 21st century," *Computer*, vol. 31, no. 3, pp. 23–30, Mar. 1998.

[10] Scripting language From Wikipedia https://en.wikipedia.org/wiki/Scripting_language

[11] JavaScript – Document Object Model or DOM http://www.tutorialspoint.com/javascript/javascript_html_dom.htm

[12] Understanding Client-side Scripting http://www.pcmag.com/article2/0,2817,1554984,00.asp

[13] Client-Side Web Development: How Scripting Languages Work https://www.upwork.com/hiring/development/how-scripting-languages-work/

[14] Introduction to server-side scripting
http://www.pythonschool.net/server-side-scripting/introduction-to-server-side-scripting/

[15] Server-side Scripting
http://www.seniornet.org/php/images/webximages/docs/Guide/pages/sss-01-intro.html

[16] Server-side Scripting: Back-End Web Development Technology
https://www.upwork.com/hiring/development/server-side-scripting-back-end-web-development-technology/

[17] Matthew Ephraim "Treating C# Like A Scripting Language"
http://mattephraim.com/blog/2009/01/02/treating-c-like-a-scripting-language/

[18] Parsing HTML documents with the Html Agility Pack
http://www.4guysfromrolla.com/articles/011211-1.aspx

[19] Easily Parse HTML Documents in C#
http://blog.olussier.net/2010/03/30/easily-parse-html-documents-in-csharp/#more-32

[20] Understanding Processing Instructions in XML
http://www.xmlplease.com/xml/xmlquotations/pi