HTML1st—A Lightweight Dynamic Web

by

Boyang Tang

Supervisor: Dr Sathiamoorthy Manoharan

BTech 451 Final Report

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

October 2016

Abstract

This report is for my final year BTech project. My project is intended to build a light-weight C# engine that handles Server-side Scripting, the engine will convert HTML pages embedded C# code fragments to pure HTML pages. This report will illustrate what tasks I have done and what open problems still exist, and discuss serval challenges that occurred throughout the year. The researches, design, implementation, and evaluations of my project are all discussed in this report.

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
October 21, 2016

Contents

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.

As we know, nowadays for making dynamic and interactive Web pages. Some powerful tools such as PHP and ASP.NET have been wildly used, but why also do we need to create another alternative approach such like HTML1st. Take PHP for instance, PHP is not light-weight, someone cannot see the contents if without running the PHP script, and it is too costly or time-consuming for people to write such a lot of script when they decide to build a large application. Additionally, there may have some security issues, since it is open sourced, so all people can see the source code, if there are bugs in the source code, it can be used by people to explore the weakness of PHP. While my task is to designed a light-weight engine that gives a convenience to people who want to make a dynamic Web, this engine saves a lot of time and avoids the security issues.

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>
```

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 21:10:04 GMT+1200
</p>
</body>
</html>
```

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 chapter 4 will describe the project implementation. After that, the testing and evaluation will be shown in chapter 5. Lastly, the chapter 6 gives a summary and conclusion of my whole year project.

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.

5

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 minimize 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 program 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.

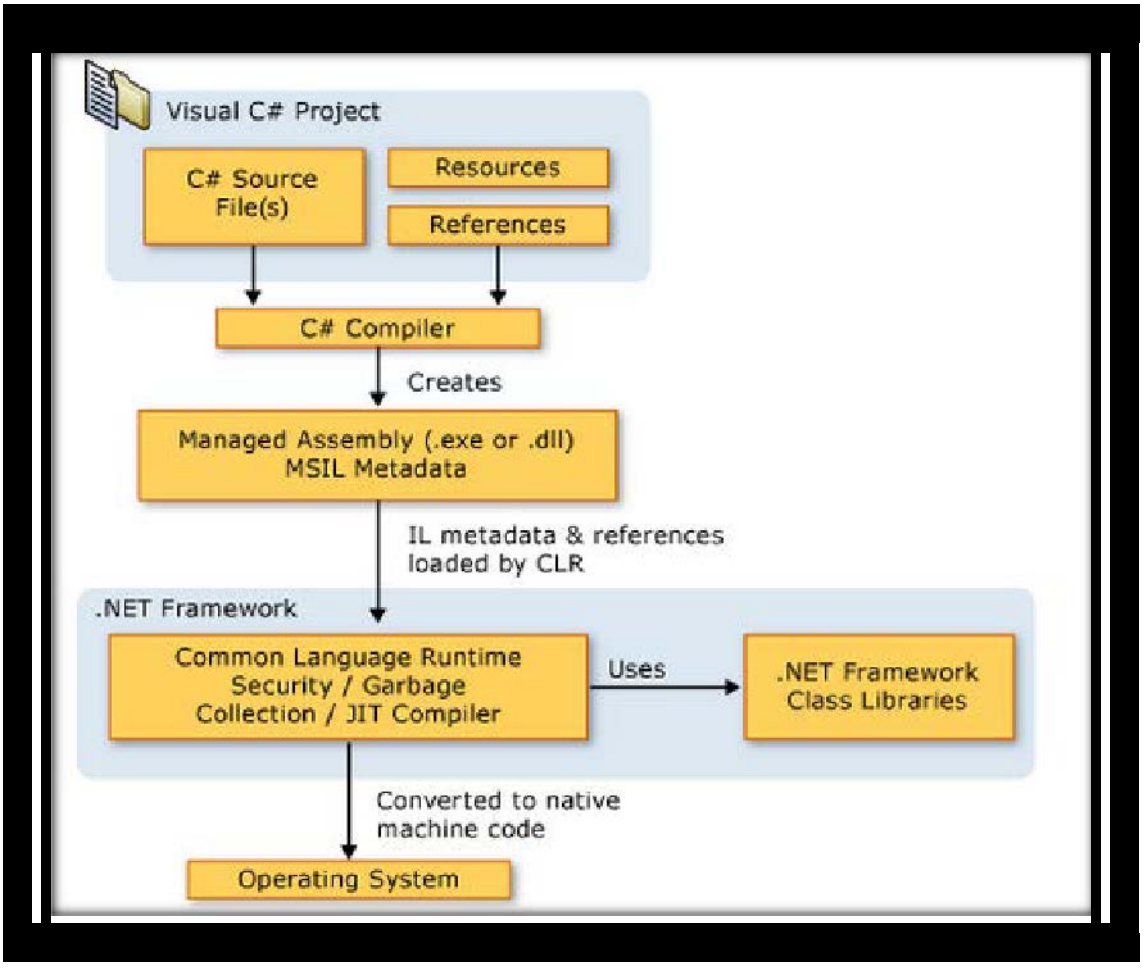


Figure 2.1: .NET Framework Platform Architectural [2]

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.

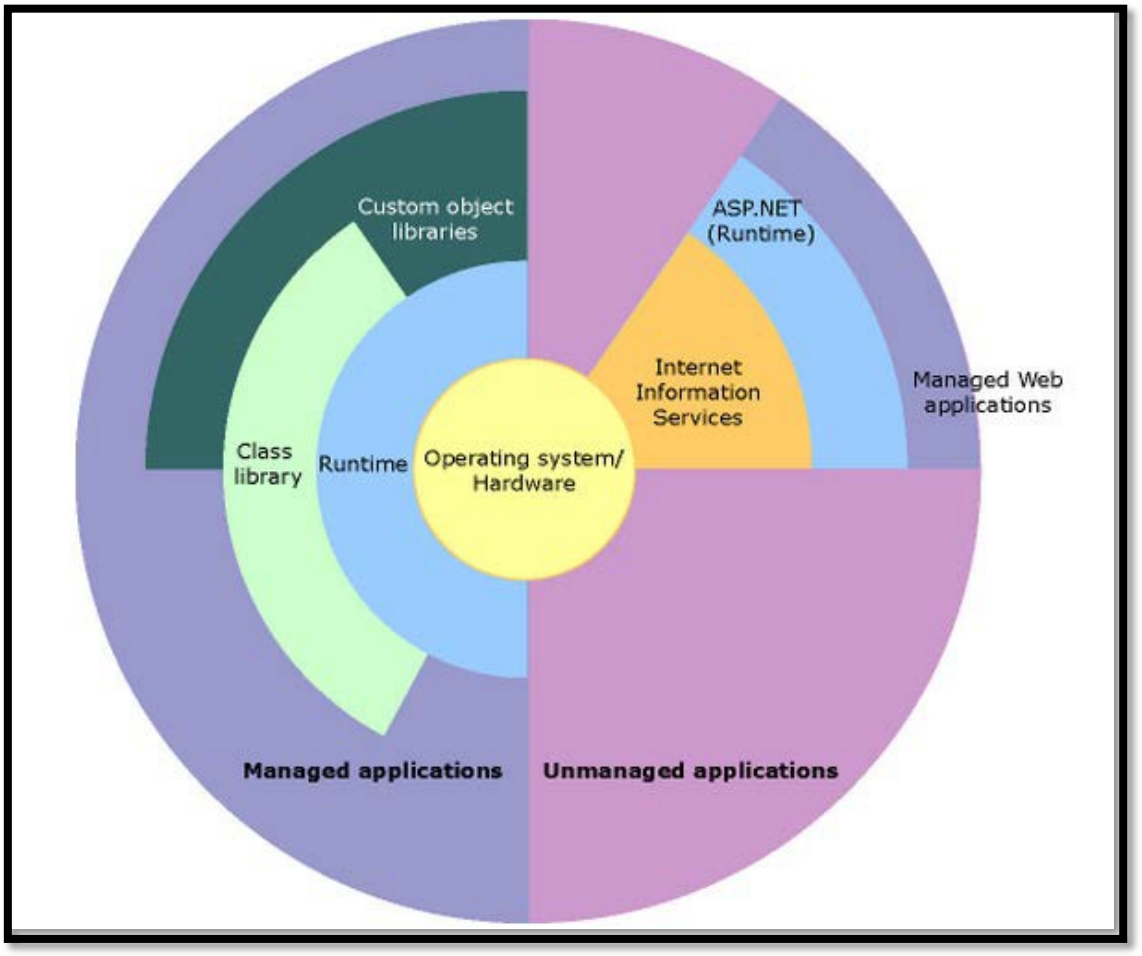


Figure 2.2: Overview of .NET Framework [4].

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 centered 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.

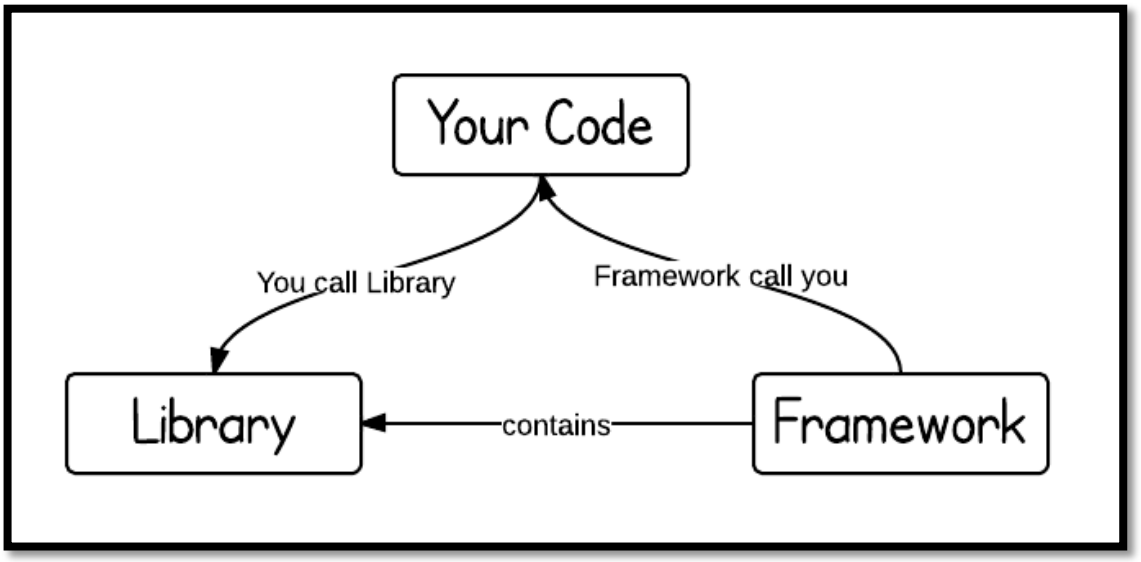


Figure 2.3: Library vs. Framework [5].

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 behaviors 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.

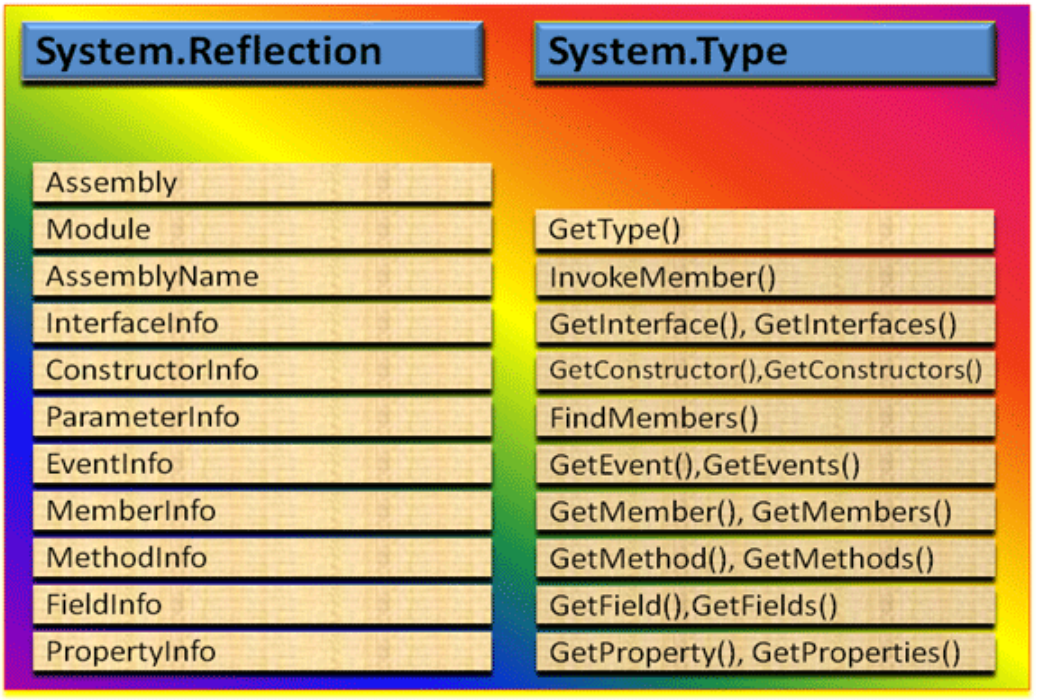


Figure 2.4: .NET Reflection Road Map [7].

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.

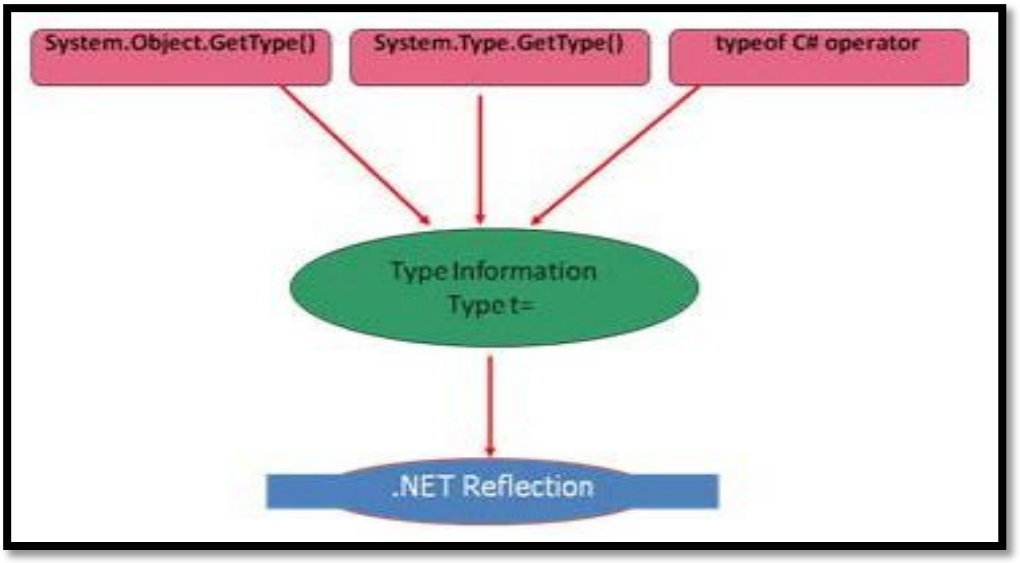


Figure 2.6: Obtaining Type Information [7].

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 2.7 shows the relationship.

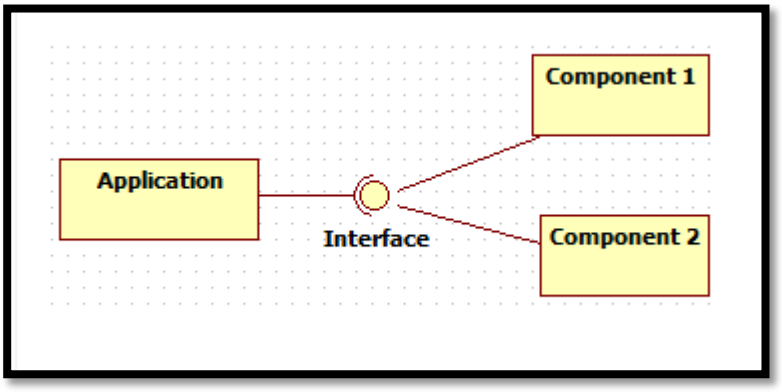


Figure 2.7: Dynamically Loading.

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.

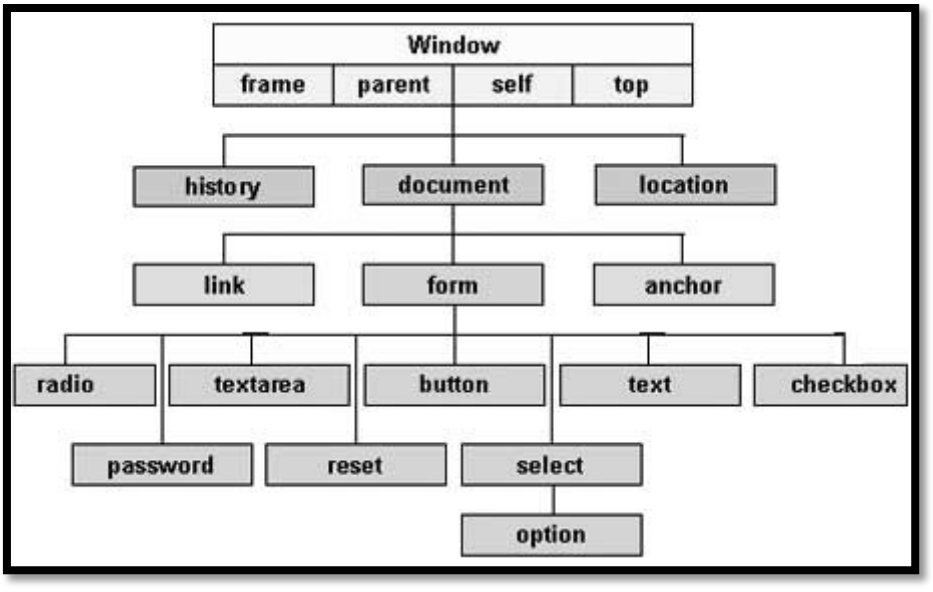


Figure 3.1: Simple hierarchy of DOM [11].

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:

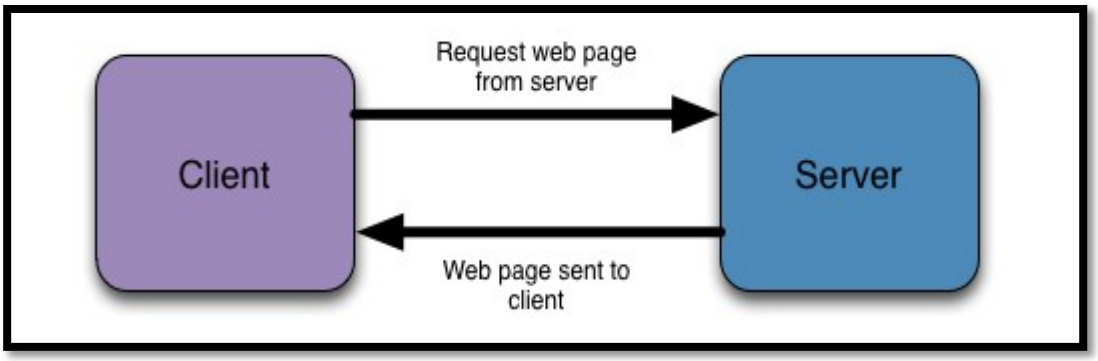


Figure 3.2: Client-Server System [14].

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:

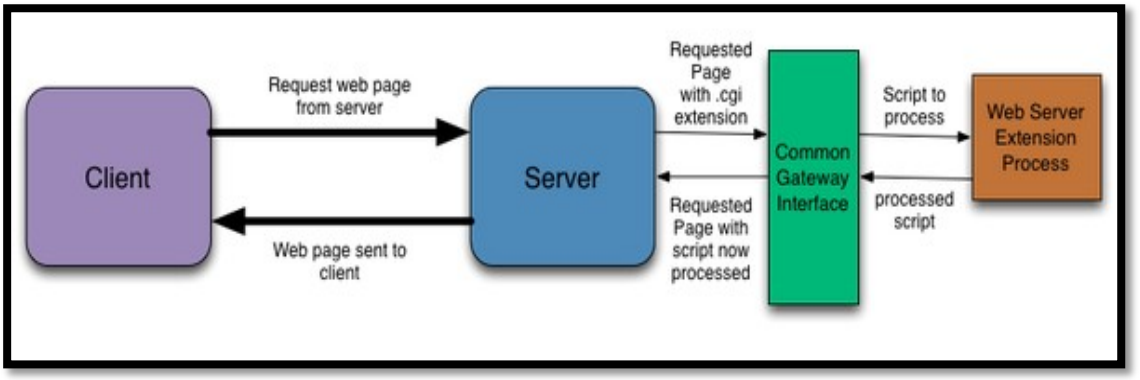


Figure 3.3: CGI with Web Server [14].

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:

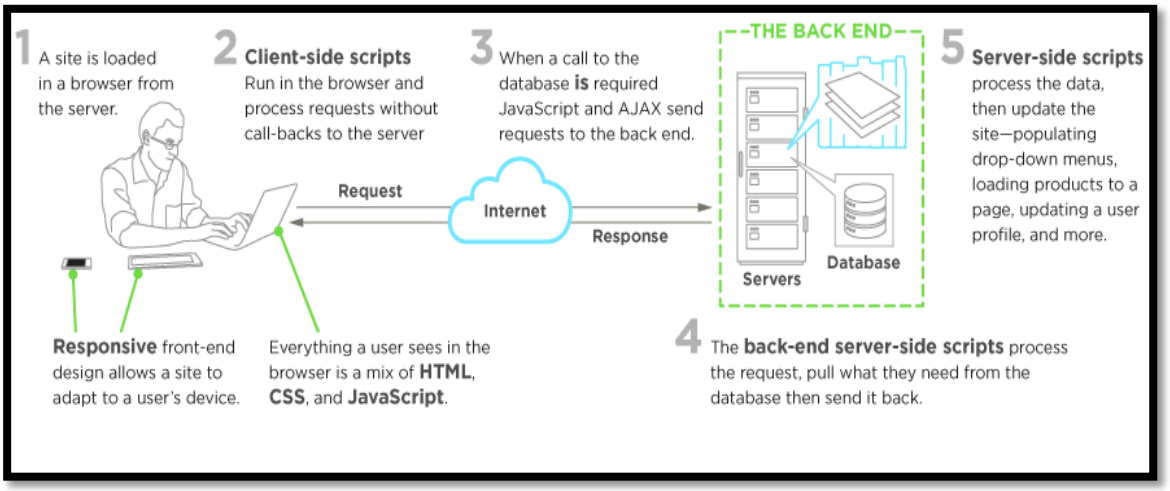


Figure 3.4:Front-End Development [13].

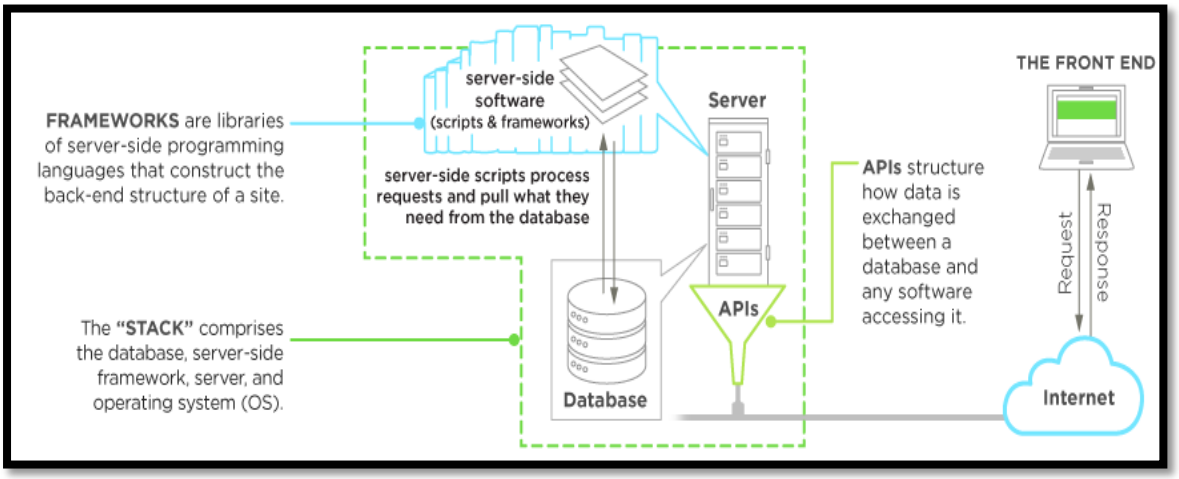


Figure 3.5: Back-End Development [16].

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()**;

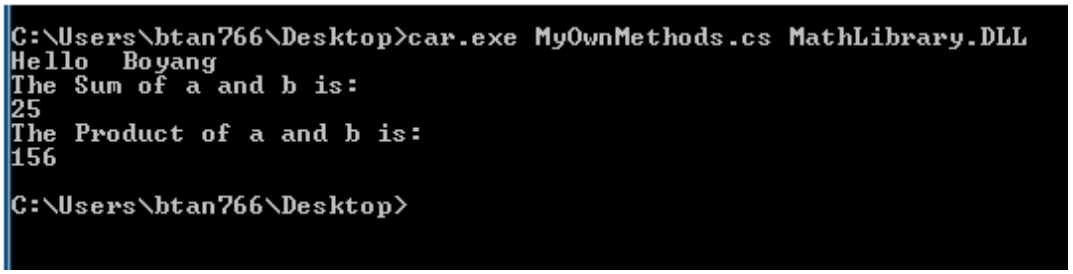
4. Compile your assembly: **CompilerResults results =provider.CompileAssemblyFromSource(parameters,SourceString);**

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:

```
  using System.Reflection;
  using System;
  using System.CodeDom;
  using Microsoft.CSharp;
  using System.CodeDom.Compiler;
  using System.Diagnostics;

class CarHTML{
static void Main(string[] args)
{
    //create a new instance of the c# compiler
    CSharpCodeProvider codeProvider = new CSharpCodeProvider();
    ICodeCompiler compiler = codeProvider.CreateCompiler();

    // Create some parameters for the compiler
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 there weren't any errors get an instance of "MyClass" and invoke
// the "Message" method on it
    if (results.Errors.Count == 0)
{
     var myMehtods = results.CompiledAssembly.CreateInstance("MyMethods");
     myMehtods.GetType().
        GetMethod("GetGreeting").
            Invoke(myMehtods, new []{ " Boyang " });


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


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

    }
}
```

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 code below shows an example for parsing with the use of HtmlAgilityPack:

```
// The HtmlWeb class is a utility class to get the HTML over HTTP
HtmlWeb htmlWeb = new HtmlWeb();

// Creates an HtmlDocument object from an URL
HtmlAgilityPack.HtmlDocument document =
htmlWeb.Load("http://www.somewebsite.com");

// 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"))
    {
      // Simple check: if the href begins with "http://", prints it out
      if (link.Attributes["href"].Value.StartsWith("http://"))
        Console.WriteLine(link.Attributes["href"].Value);
    }
  }
}
```

A more efficient way with the use of XPath expressions:

```
// The HtmlWeb class is a utility class to get the HTML over HTTP
HtmlWeb htmlWeb = new HtmlWeb();

// Creates an HtmlDocument object from an URL
HtmlAgilityPack.HtmlDocument document =
htmlWeb.Load("http://www.somewebsite.com");

// 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);
}
```

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 code 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>
```

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.

3.3.3 Second plan

As I mentioned before, the HTML Agility Pack does not provide an ability to extract C# fragment codes with format of Processing Instruction (PI) from a HTML file. There may have two alternative ways to resolve this issue, which are creating an own HTML parser based on the requirements given by my project, or introducing a new tag like <sourceCode> into HTML file instead of enclosing C# fragments within <? ?>, then, the new format looks like < sourceCode > C# fragment < /sourceCode >.

However, considering the first choice is time consuming and cannot guarantee the stability of its own, we prefer to take the second solution. Thereafter, each C# fragment or function is enclosed within a new introduced element, we can also add some attributes to these new elements, and each attribute is made of attribute name and value. This gives HTML parser a convenience that by using specified attribute values to extract the specific code fragments. Note that we always quote attribute values with double quotes, such like < sourceCode **class =** "html1st" > function(); < /sourceCode >, where the attribute name is "class" and its value is equal to "html1st".

In this case, there is another issue may occur during the parsing step, some special characters need to be escaped in HTML (e.g. <, >, &) and therefore if they appear in code fragments, they need to be escaped too. This issue occurs in such condition, say, <div **class**="html1st"> Print(100 >> 1); , there is a symbol " > " within it, it may mix-up the view of HTML parser, and errors will appear. This means HTML must sometimes be encoded, the special characters such as "<", ">", "&" can be converted to "<", ">", and "&" respectively. Then, it can be presented as <div **class**="html1st"> Print(100 > > 1); . During the HTML file creation process, we can directly write them in the format of above, it promises future stability in the period of HTML parsing. However, these symbols in 'code' cannot be figured out by C# language, therefore, before you can execute the code, you need to reverse those changes, with WebUtility.HtmlDecode methods in the C# language, we do this without writing any custom code. In addition, another fix might be to move such code outside the HTML into the code's own library (i.e., DLL) and link in the DLL at compile time.

For example:
<source>PrintResult();</source>
Where PrintResult() { bool ok = 2 < 1; System.Console.WriteLine(ok); } is written in the library code.

The total look of a simple input HTML file is following:

```
<html>
<head><title> This is my project HTML1st </title></head>
<body>
<!--  How we do function calls is to put them within new tag <div> -->
<h1><div class="html1st">GetGreeting("Boyang");</div></h1>
<p>
Here is how I would greet you:
<div class="html1st">GetGreeting("Boyang");</div>
</p>
<p>
This should print out:
<div class="html1st">GetNumber("12","13");</div>
</p>
</body>
</html>
```

Chapter 4

4. Implementation

This section will merge all steps that I have already studied and finished, then integrate them into a real application. Details about code and problems I encountered when I was programming will be explained step by step. Lastly, this part will give a summary and overview of the project code.

4.1 External Libraries

Before we set up the implementation, giving a recall to my project goal, my project is to build a C# engine that can parse a HTML file in which is embedded with some C# fragments code, and then convert this kind of file to a pure HTML file. I assume the engine than can efficiently parse real-world HTML and extract code fragments should be looked as a main success achieved by my project. Considering the application should be light-weight and developing a new HTML parser is time-consuming, we decided to rely on a third-party library which is already commonly used for parsing HTML.

I tried to make the most of internal libraries which are offered by .NET Framework. The second plan has been introduced in the last part of former section, it totally depended on HTML Agility Pack. I only used this external library because it enabled my program to parse real-world HTML that could not be done by standard .NET libraries. This packet produces some significant benefits, to be more specific, HTML Agility Pack is an agile HTML parser that builds a read/write DOM and supports plain XPath, which means it allows you to return or retrieve specified nodes through the use of XPath expressions. It is a .NET code library that allows you to parse "out of the web" HTML files. The parser is very tolerant with "real world" malformed HTML. The object model is very similar to what proposes System.Xml, but for HTML documents (or streams).

4.2 Parsing HTML

This part shows how to get started with HTML Agility Pack and then use it to parse HTML files. Some samples with HTML input and output will be listed in the end of this part to see how parsing can be achieved by using this packet in .NET. Firstly, in order to get HTML Agility Pack in my application, I can install it in my project by running the following command in the Package Manager Console:

```
PM> Install-Package HtmlAgilityPack
```

In all cases, you can open the Console in Visual Studio through the **Tools** > **NuGet Package Manager** > **Package Manager Console** command.

After adding the library via Nuget, we are still missing a using directive or an assembly reference, then, we need to include the reference in my program page using command as follows:

```
using HtmlAgilityPack;
```

There are two situations when we loading a HTML, it could be loaded as a page from Internet, or it could be loaded directly from a saved document. In my project, we attempt to parse HTML files which are already created by someone, so we did the code like following:

```
HtmlDocument html = new HtmlDocument();
html.Load("sampleInputHtml.html");
var root = html.DocumentNode;
var commonPosts = root.SelectNodes("//div[@class = 'html1st']");
```

As so far, we have loaded the entire HTML file into the object html, once the loaded HTML document is ready to be parsed, using the `DocumentNode` property of the `HtmlDocument` to return the root element of HTML, thereafter, we can further filter my search by specifying elements that have certain conditions. In this case, we want to select all elements with a class of “html1st”. This code will search all div with the attribute “class” is equal to “html1st” from the page and return in nodes.

After that, we collect a set of nodes that contain those specific HTML elements, but the most useful stuff that for further work are embedded code functions within element tags. It is essential to wipe off all kinds of such tags or other characters and only leave the C# code fragments. As I mentioned in the second plan, some symbols such as “<”, “>”, and “&” should be escaped in HTML, thereby writing them as “<”, “>”, and “&” instead. Before you can execute the code, you need to reverse those changes, as those re-written symbols in ‘code’ cannot be figured out by C# language. The WebUtility.HtmlDecode method in the C# language enables to fix this issue without writing any further custom code. This is shown as following code:

```
ArrayList list = new ArrayList();
foreach (var htmlNode in commonPosts)
            {
                string b = WebUtility.HtmlDecode(htmlNode.InnerText);
                list.Add(b);

            }
```

Where the InnerText is used for extracting code fragments within nodes and I also create an ArrayList to save them.

4.3 Using the compiled code

After parsing HTML pages successfully, we need to execute the methods or functions that have been extracted from those documents. At this point, my program does not have access to a .NET assembly at compile time but I want to run those code in it. Reflection in .NET makes it possible to dynamically load an assembly and run code in it without early access. So, I created a C# class file named “MyOwnMethods” which contains implementations of all customer methods that refer to extracted code fragments. In addition, I also created a C# class library project named “MathLibrary” which includes some other useful methods that would be called by methods in above class. Then, I putted “MyOwnMethods.cs” and “MathLibrary.DLL” into the *Debug* folder within the *bin* folder of *Visual Studio* for the user to reference those assemblies and invoke a method of a type at runtime. Firstly, we can have a look at my solution for the project, the code is shown as follows:

```
//create a new instance of the C# compiler
CSharpCodeProvider compiler = new CSharpCodeProvider();
// Create some parameters for the compiler
CompilerParameters parameters = new CompilerParameters();

parameters.GenerateExecutable = false;
parameters.GenerateInMemory = true;

parameters.ReferencedAssemblies.Add("System.dll");
parameters.ReferencedAssemblies.Add("MathLibrary.DLL");
var results =
compiler.CompileAssemblyFromFile(parameters,"MyOwnMethods.cs");
```

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

  ArrayList replace = new ArrayList();
  int count = 0;

  foreach (string i in list)
     {

       int p = i.IndexOf('(');
       int q = i.IndexOf(')');

      string nameOfMethod = i.Substring(0, p);
      string b = i.Substring(p + 1, q - p - 1);
      string words = b.Replace("\"", "");
      string[] parameter = words.Split(',');


   if (parameter.Length <= 1){
   var myMehtod = results.CompiledAssembly.CreateInstance("MyMethods");
                  var a = myMehtod.GetType().
                        GetMethod(nameOfMethod).
                        Invoke(myMehtod, new[] { parameter[0] });
                  replace.Add(a);
              }

      if (parameter.Length > 1){
      var math = results.CompiledAssembly.CreateInstance("MyMethods");
                  var par = math.GetType().
                      GetMethod(nameOfMethod).
                      Invoke(math, new[] { parameter });
                  replace.Add(par);
              }

          }
```

To programmatically compile my code I need to create a C# compiler, the compiler created by using an instance of **CSharpCodeProvider** class. Then I created parameters of the compiler by using an instance of **CompilerParameters** class, which contains a set of parameters that would be passed to compiler. In my project, those parameters are some reference assemblies like dynamic link libraries. In order to execute embedded functions extracted from HTML, firstly, we can directly compile the code from a source file by using **CompileAssemblyFromSource** method.

Once my code was compiled into an assembly, I could use that assembly to create instances of classes from my source code, then, using reflection to invoke a method which is corresponding to that of methods list (the ArrayList I have mentioned before, it was used to collect all extracted methods). Note that I tried to split up each method into two parts, one of them was name of a method, and another was parameters included in that method. This gave a great convenience to invoke a specified method. By then, I created another list to collect the outputs of running those code fragments thereby using these return values to replace embedded fragments in HTML pages.

4.4 Producing pure HTML file

The last step is to replace all C# code fragments within HTML file, which means my engine enables to re-edit HTML and replace certain texts inside it. So the idea is having a HTML template with some certain texts that are going to be replaced by other texts, in this project, the certain text are the inner texts of some specified HTML tags. In the former steps, I had collected a set of inner texts which are C# code fragments, and I also collected a list of return values that are the output of running these fragments. I found HTML Agility Pack did give an ability to replace those fragments by return values, and generate a pure HTML file in the end. Before we started, I made a copy of sample input HTML file to another file, then we could reload this new file and modify the inner texts inside it, this made no changes of the original one, so we could make a comparison of them after replacing. The following code shows this action:

```
using (StreamReader stream = new StreamReader("sampleInputHtml.html"))
using (StreamWriter writeStream = new
StreamWriter("sampleOutputHTML.html"))
                    {
                        string line;
                        while ((line = stream.ReadLine())!= null)
                        {
                            writeStream.WriteLine(line);
                        }
                      }
```

When I was trying to use HTML Agility Pack (with XPath expression) to replace the InnerText of some tags, my program occurred an exception, and what I found is that InnerText is read only, it cannot be modified directly. Fortunately, I found a simple way to fix this issue, this is shown as following:

```
    HtmlDocument html2 = new HtmlDocument();
    html2.Load("sampleOutputHTML.html");
    var root2 = html2.DocumentNode;
    var countNum = root2.SelectNodes("//div[@class =
'html1st']//text()");

    foreach (var htmlNode in countNum)
                {
                    var newNodeStr = replace[count];
                    var newNode = HtmlNode.CreateNode( ""+newNodeStr+""
);
                    htmlNode.ParentNode.ReplaceChild(newNode, htmlNode);
                    count++;
                    html2.Save("sampleOutputHTML.html");
                }
```

The expression is used here: `//div[@class = 'html1st']` selects the `div`, which is not an HtmlTextNode, the countNum variable holds null, therefore, exception will occur. `HtmlTextNode` has a property `Text` which could be used to set the necessary value. But before this you should get that text node. This could be easily done with this expression: `//div[@class = 'html1st']//text()`. Lastly, I created a new node without changing tags and attributes, meanwhile the InnerText had been replaced by a new certain text. And using the new node to replace the original one, the modifications inside HTML were saved.

4.5 Optimization Design

Up to now my program totally satisfies the requirements of this project, the HTML1st engine can parse real-world HTML with some embedded C# code fragments, and can extract those fragments thereby compiling them on fly, in the meantime, using reflection to invoke each method at runtime. Thereafter, HTML1st can replace those C# fragments with the return values.

The only drawback to this approach is that my program depends on a third party library. Although the HTML Agility Pack is brilliant, it gives a lot of conveniences and remains stable when parsing real-world malformed HTML. But we still have to consider the self-independence of my program, since introducing an external dependency sometimes brings with it a legal security risk. Beyond that, many times the extensive features of external libraries become too large to test properly and they add cognitive overhead to a project by requiring new developers to understand additional knowledge. Let us suppose that the HTML1st would be wildly used by other developers in the future, it seems to be better and safer if we base our work on existing standard library packages.

However, as previously mentioned, the embedded C# code fragments were enclosed within <? ?> and regarded as Processing Instructions (PIs). In that case, XML parser could not parse an HTML as a XML, because it had no ability to identify those PIs. But, after introducing the second plan, the formal semantics of an HTML do not change, it is possible to use XML parser with XPath expression to parse an HTML page. The optimization of this aspect allows my program to perform as well as before without deploying any external libraries, since .NET Framework provides an XmlDocument class, along with XMLElement, which allows you to access data by calling DOM methods or to use XPath navigation to query information in the DOM. You can use XPath to find a single, specific node or to find all nodes that match some criteria. In my program, I only need to modify the code where that part is used to parsing HTML. The modifications have been shown as following:

```
//Parsing sampleInput Html
XmlDocument doc = new XmlDocument();
doc.Load("sampleInputHtml.html");
XmlNode root = doc.DocumentElement;
XmlNodeList commonPosts = root.SelectNodes("//div[@class = 'html1st']");

//Parsing sampleOutput Html which is copied from sampleInput Html
XmlDocument doc2 = new XmlDocument();
doc2.Load("sampleOutputHTML.html");
XmlNode root2 = doc2.DocumentElement;
XmlNodeList countNum = root2.SelectNodes("//div[@class = 'html1st']");
```

Additionally, it is necessary to amend the part that is used for replacing:

```
foreach (XmlNode XmlNode in countNum)
            {
                XmlAttribute attribute = XmlNode.Attributes[0];
                XmlElement newNode = doc2.CreateElement("div");
                newNode.SetAttribute(attribute.Name,attribute.Value);
                newNode.InnerText = (string)replace[count];
                XmlNode.ParentNode.ReplaceChild(newNode, XmlNode);
                count++;
                doc2.Save("sampleOutputHTML.html");
            }
```

The complete source code of this project is shown in Appendix.

Chapter 5

5. Evaluation

This chapter illustrates a practice examination for my project application

5.1 Testing

Basically, I want to ensure that what I create does what it is supposed to do, it is very important to see whether the quality of my application is good or not. Moreover, it is also necessary to ensure that the HTML1st engine should not result into any failures. In order to evaluate efficiency and stability of my project, I have written a set of test cases:

Test case 1: give a greeting and do some calculations

sampleInputHtml.html:

```
<html>
<head><title>Test case 1</title></head>
<body>
<!--  How we do function calls is to put them within a new
tag <div>
-->
<h1><div class="html1st">GetGreeting("Boyang");</div></h1>
<p>
Here is how I would greet you:
<div class="html1st">GetGreeting("Boyang");</div>
</p>
<p>
This should print out:
<div class="html1st">GetNumber("12","13");</div>
</p>
</body>
</html>
```

sampleOutputHtml.html:

```
<html>
  <head>
    <title>Test case 1</title>
  </head>
  <body>
    <!--  How we do function calls is to put them within a new
tag <div> -->
    <h1>
    <div class="html1st">Hello Boyang</div>
    </h1>
    <p>
    Here is how I would greet you: <div class="html1st">Hello
    Boyang</div>
    </p>
    <p>
    This should print out: <div class="html1st">The Sum of a
    and b is:25 The Product of a and b is:156</div>
    </p>
  </body>
</html>
```

We can see the input HTML and the output HTML are exactly consistent without changes of format, the markups and all contents are same, expect the embedded C# code fragments have been replaced by return values of running those code fragments.

Test case 2: Random numbers

sampleInputHtml.html:

```
<html>
<head><title>Test case 2</title></head>
<body>
<!--  How we do function calls is to put them within a new tag
<div>  -->
<h1>Any random number is:
<div class="foo">Random("6","10");</div>
</h1>
<p>
The sum of random numbers is:
<div class="foo">Sum("6","10");</div>
</p>
<p>
The remainder after division of random numbers:
<div class="foo">Modulo("6","10");</div>
</p>
</body>
</html>
```

sampleOutputHtml.html:

```
<html>
  <head>
    <title>Test case 2</title>
  </head>
  <body>
    <!--  How we do function calls is to put them within a new tag
   <div>  -->
   <h1>Any random number is:
   <div class="foo">random num between 6 and 10 is 6</div>
   </h1>
   <p>
   The sum of random numbers is:
   <div class="foo">the two random nums are 6 and 7, and the sum
   of them is 13</div>
   </p>
        <p>
   The remainder after division of random numbers:
   <div class="foo">the two random nums are 6 and 7, and the
   remainder after division is 6</div>
   </p>
  </body>
</html>
```

In this test case, I want to see if I could choose a random number between 6 and 10, next, randomly generating two numbers between 6 and 10, and calculating the sum of them also the remainder of division. The result shows the HTML1st engine successfully accomplishes the goal of my project.

Test case 3: Find factors of a random number

sampleInputHtml.html:

```
<html>
<head><title>Test case 3</title></head>
<body>
<!--  How we do function calls is to put them within a new tag
<div>  -->
<h1>Any random number between 6 and 10 is:
<div class="num">Random("6","10");</div>
</h1>
<p>
Find factors of a random number between 6 and 20:
<div class="num">Factors("6","20");</div>
</p>
<p>
Find factors of a random number between 20 and 50:
<div class="num">Factors("20","50");</div>
</p>
</body>
</html>
```

sampleOutputHtml.html:

```
<html>
  <head>
    <title>Test case 3</title>
  </head>
  <body>
    <!--  How we do function calls is to put them within a new tag
   <div>  -->
   <h1>
   Any random number between 6 and 10 is:
   <div class="num">random num between 6 and 10 is 9</div>
   </h1>
    <p>
   Find factors of a random number between 6 and 20:
   <div class="num">random num is 19, and the factors of this
   number are 1,19</div>
   </p>
    <p>
   Find factors of a random number between 20 and 50:
   <div class="num">random num is 48, and the factors of this
   number are 1,48,2,24,3,16,4,12,6,8</div>
   </p>
  </body>
</html>
```

In this test case, I give a more complex example to test the accuracy of my application, the result shows the success as same as before.

Test case 4: Testing whether HTML1st can handle special symbols

sampleInputHtml.html:

```
<html>
<head><title>Test case 4</title></head>
<body>
<!--  How we do function calls is to put them within a new tag
<div>-->
<h1>Boolean function:
<div class="bool">printOut("&gt; and &lt;");</div>
</h1>
<p>
The result is: <div class="bool">bigger("6&gt;7");</div>
</p>
<p>
The result is: <div class="bool">smaller("8&lt;3");</div>
</p>
</body>
</html>
```

sampleOutputHtml.html:

```
<html>
  <head>
    <title>Test case 4</title>
  </head>
  <body>
    <!--  How we do function calls is to put them within a new tag
    <div>-->
   <h1>Boolean function:
    <div class="bool">testing special symbols &gt; and &lt; in
    HTML</div>
    </h1>
     <p>
    The result is: <div class="bool">6&gt;7 is false</div>
  </p>
     <p>
    The result is: <div class="bool">8&lt;3 is false</div>
  </p>
  </body>
</html>
```

An Issue - Some characters need to escaped in HTML (e.g. <, >, &) and therefore if they appear in the ‘code’, they need to be escaped too. So when someone creates an HTML file, he or she can edit above symbols as “<”, “>”, and “&” instead. However, before you can execute the code, you need to reverse these changes. This test case shows the HTML1st engine can easily handle the above issue.

Chapter 6

Conclusion

This section will give a conclusion for my report and conclude with a summary of strengthens and weaknesses of my project.

This project was given to me by Computer Science Department of UoA, and my task was to develop a light-weight C# engine that handles server-side scripting. The engine will convert HTML pages with embedded C# code fragments to pure HTML pages. This report covers the entire work of my final year project, of which the studies, researches, design, implementation, and evaluations were all discussed in the former sections of my report.

I started by learning C# .NET programming language and then I did some researches about functionalities given by Reflection in .NET Framework. After having some extent knowledge about scripting language, I created a small program and used it to load an assembly that contains some C# code, and then executed them as part of my program, it did invoke a type's method at runtime. The biggest challenge in my project was that how to properly parse a real-world HTML page with embedded C# code fragments. At the beginning of that phase, my program was basically relied on a third party library HTML Agility Pack. I putted code fragments within <? ?> and regarded them as Processing Instructions (PIs), but this package could not figure out PIs. In the second semester, we decided to try the second plan, which was introducing a new tag into HTML pages, and each code fragment was enclosed within a new tag instead of PI. After that, I integrated all finished segment into a real application, and my program did a well job without errors. It could extract out of all fragments and dynamically loaded an assembly which was corresponding to those code fragments thereby compiling them on fly, in the meantime, using reflection to invoke each method at runtime. Thereafter, HTML1st could replace those C# fragments with the return values.

But, considering this C# engine should be self-independent, I assume that it would be better and safer if we base our work on existing standard library packages. I found that it is possible to use XML parser with XPath expression to parse an HTML page without deploying any external libraries, this version of my program (without HTML Agility Pack) performed as well as before. I wrote a set of test cases for both versions, and results gave a proof that my program exactly did what it was supposed to do. Both of them worked out the task of my project with no failures. Also, both versions can handle the issue such as some characters need to escaped in HTML (e.g. <, >, &).

Personally, I think the HTML1st is potential to be wildly used in the real world or used as a foundation of other developments, since this C# engine is efficient and light-weight. From then on, people no longer need to write too much scripts to tell server how and what contents should be rendered into a human viewable website, because this is too costly or time-consuming. But now it is possible to directly design a HTML page by using standard HTML format, and just put some purposive actions as code fragments in an HTML page. The HTML1st C# engine will handle those server-side scripting and then convert HTML pages with embedded C# code fragments to pure HTML pages.

On the other hand, the accuracy and effectiveness of this engine to fulfil the task may be limited by the lack of realistic work for large scale experimentation. These days most websites on the Internet have dynamic content. In order to deliver dynamic content the volume and variety of C# code fragments within HTML pages may increase significantly. It is not possible to write all implementations of those fragments in one assembly, then we would reference other assemblies at runtime, this can be achieved by using dynamic link libraries (DLL). This part has been completed in my project, but I believe there needs more tests in the further work.

In conclusion, this project was successfully completed on time, while my program does what exactly it is supposed to do, there may also have some other drawbacks or bugs which are not mentioned and can be improved in the future.

Appendixes

Source code Version 1 (without HTML Agility Pack):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Collections;
using Microsoft.CSharp;
using System.CodeDom.Compiler;
using System.Reflection;
using System.IO;
using System.Text.RegularExpressions;
using System.Web;
using System.Xml;

namespace Final
{
    class Program
    {
        static void Main(string[] args)
        {

            //Parsing sampleInput Html
            XmlDocument doc = new XmlDocument();
            doc.Load("sampleInputHtml.html");
            XmlNode root = doc.DocumentElement;
            XmlNodeList commonPosts = root.SelectNodes("//div[@class = 'html1st']");
            ArrayList list = new ArrayList();

            foreach (XmlNode a in commonPosts)
            {
                string b = WebUtility.HtmlDecode(a.InnerText);


                list.Add(b);


            }

            //create a new instance of the c# compiler
            CSharpCodeProvider compiler = new CSharpCodeProvider();


            // Create some parameters for the compiler
            CompilerParameters parameters = new CompilerParameters();
            parameters.GenerateExecutable = false;
            parameters.GenerateInMemory = true;

            parameters.ReferencedAssemblies.Add("System.dll");
            parameters.ReferencedAssemblies.Add("MathLibrary.DLL");
            var results =
            compiler.CompileAssemblyFromFile(parameters,"MyOwnMethods.cs");
```

```
            if (results.Errors.Count == 0)
              {
                  ArrayList replace = new ArrayList();
                  int count = 0;
                  foreach (string i in list)
                  {

                      int p = i.IndexOf('(');
                      int q = i.IndexOf(')');
                      string nameOfMethod = i.Substring(0, p);


                      string b = i.Substring(p + 1, q - p - 1);
                      string words = b.Replace("\"", "");
                      string[] parameter = words.Split(',');


                      if (parameter.Length <= 1)
                       {
                          var myMehtod =
                          results.CompiledAssembly.CreateInstance("MyMethods");
                          var a = myMehtod.GetType().
                                GetMethod(nameOfMethod).
                                Invoke(myMehtod, new[] { parameter[0] });

                          replace.Add(a);
                      }
                       if (parameter.Length > 1)
                       {
                          var math =
                          results.CompiledAssembly.CreateInstance("MyMethods");
                          var par = math.GetType().
                              GetMethod(nameOfMethod).
                              Invoke(math, new[] { parameter });

                          replace.Add(par);
                      }

                  }

                using (StreamReader stream = new StreamReader("sampleInputHtml.html"))
                using (StreamWriter writeStream = new
                                              StreamWriter("sampleOutputHTML.html"))
                      {
                          string line;

                          while ((line = stream.ReadLine())!= null)
                          {
                              writeStream.WriteLine(line);
                          }
                      }


                   //Parsing sampleOutput Html which is copied from sampleInput Html
                  XmlDocument doc2 = new XmlDocument();
                  doc2.Load("sampleOutputHTML.html");
                  XmlNode root2 = doc2.DocumentElement;
                  XmlNodeList countNum = root2.SelectNodes("//div[@class =
'html11st']");
```

```
                foreach (XmlNode XmlNode in countNum)
                {


                    XmlAttribute attribute = XmlNode.Attributes[0];
                    XmlElement newNode = doc2.CreateElement("div");
                    newNode.SetAttribute(attribute.Name,attribute.Value);
                    newNode.InnerText = (string)replace[count];
                    XmlNode.ParentNode.ReplaceChild(newNode, XmlNode);
                    count++;
                    doc2.Save("sampleOutputHTML.html");
                }
            }



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



            Console.Read();


        }
    }
}
```

Source code Version 2 (with HTML Agility Pack):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using HtmlAgilityPack;
using System.Net;
using System.Collections;
using Microsoft.CSharp;
using System.CodeDom.Compiler;
using System.Reflection;
using System.IO;
using System.Text.RegularExpressions;
using System.Web;
using System.Xml;

namespace Final
{
    class Program
    {
        static void Main(string[] args)
        {
             //Parsing sampleInput Html
             HtmlDocument html = new HtmlDocument();
             html.Load("sampleInputHtml.html");
             var root = html.DocumentNode;
             var commonPosts = root.Descendants().Where(n =>
                 n.GetAttributeValue("class", "").Equals("html1st"));


            ArrayList list = new ArrayList();

            foreach (var a in commonPosts)
            {
                string b = WebUtility.HtmlDecode(a.InnerText);


                list.Add(b);


            }

            //create a new instance of the c# compiler
            CSharpCodeProvider compiler = new CSharpCodeProvider();

            // Create some parameters for the compiler
            CompilerParameters parameters = new CompilerParameters();
            parameters.GenerateExecutable = false;
            parameters.GenerateInMemory = true;

            parameters.ReferencedAssemblies.Add("System.dll");
            parameters.ReferencedAssemblies.Add("MathLibrary.DLL");
            var results =
            compiler.CompileAssemblyFromFile(parameters, "MyOwnMethods.cs");
```

```
if (results.Errors.Count == 0)
     {
         ArrayList replace = new ArrayList();
         int count = 0;

         foreach (string i in list)
         {

             int p = i.IndexOf('(');
             int q = i.IndexOf(')');
             string nameOfMethod = i.Substring(0, p);


             string b = i.Substring(p + 1, q - p - 1);
             string words = b.Replace("\"", "");
             string[] parameter = words.Split(',');

             if (parameter.Length <= 1)
             {
         var myMehtod = results.CompiledAssembly.CreateInstance("MyMethods");
         var a = myMehtod.GetType().
                   GetMethod(nameOfMethod).
                    Invoke(myMehtod, new[] { parameter[0] });

                 replace.Add(a);
             }

             if (parameter.Length > 1)
             {
             var math = results.CompiledAssembly.CreateInstance("MyMethods");
             var par = math.GetType().
                     GetMethod(nameOfMethod).
                     Invoke(math, new[] { parameter });

                 replace.Add(par);
             }

         }
 using (StreamReader stream = new StreamReader("sampleInputHtml.html"))
 using (StreamWriter writeStream = new StreamWriter("sampleOutputHTML.html"))
             {
                 string line;


                 while ((line = stream.ReadLine())!= null)
                 {
                     writeStream.WriteLine(line);
                 }
             }
```

```
            //Parsing sampleOutput Html which is copied from sampleInput Html
            HtmlDocument html2 = new HtmlDocument();
            html2.Load("sampleOutputHTML.html");
            var root2 = html2.DocumentNode;
            var countNum = root2.SelectNodes("//div[@class = 'html1st']//text()");



                foreach (var htmlNode in countNum)
                {
                    var newNodeStr = replace[count];
                    var newNode = HtmlNode.CreateNode( ""+newNodeStr+"" );
                    htmlNode.ParentNode.ReplaceChild(newNode, htmlNode);
                    count++;
                    html2.Save("sampleOutputHTML.html");
                }


            }

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



            Console.Read();


        }
    }
}
```

Reference assembly (MyOwnMethods.cs):

```
using System;
using System.Collections.Generic;
using UtilityMethods;
class MyMethods
{


    public string GetGreeting(string name)
    {
        string a = "Hello" + " " + name;
        return a;
    }

    public string printOut(string a)
    {

        string s = "testing special symbols " + a + " in HTML";
        return s;
    }


    public string GetNumber(string[] a)
    {
        long num1 = long.Parse(a[0]);
        long num2 = long.Parse(a[1]);

        string c = "The Sum of a and b is:" + (num1 + num2) + " The Product of a and
                    b is:" + (num1 * num2);
        return c;
    }


    public string Random(string[] a)
    {
        int num1 = Int32.Parse(a[0]);
        int num2 = Int32.Parse(a[1]);
        Random rnd = new Random();
        int one = rnd.Next(num1, num2);
        string s = "random num between " + num1 + " and " + num2 + " is " + one;


        return s;
    }

    public string Sum(string[] a)
    {
        int num1 = Int32.Parse(a[0]);
        int num2 = Int32.Parse(a[1]);
        Random rnd = new Random();
        int one = rnd.Next(num1, num2);
        int two = rnd.Next(num1, num2);

        int c = one + two;
        string s = "the two random nums are " + one + " and " + two +
                   ", and the sum of them is " + c;
        return s;
    }
```

```
public string Modulo(string[] a)
{
    int num1 = Int32.Parse(a[0]);
    int num2 = Int32.Parse(a[1]);
    Random rnd = new Random();
    int one = rnd.Next(num1, num2);
    int two = rnd.Next(num1, num2);

    int c = one % two;
    string s = "the two random nums are " + one + " and " + two +
              ", and the remainder after division is " + c;
    return s;

}
```

```
public string Factors(string[] a)
{
    int num1 = Int32.Parse(a[0]);
    int num2 = Int32.Parse(a[1]);
    Random rnd = new Random();
    int number = rnd.Next(num1, num2);

    List<int> factors = new List<int>();
    int max = (int)Math.Sqrt(number);  //round down
    for (int factor = 1; factor <= max; ++factor)
    {
        //test from 1 to the square root, or the int below it, inclusive.
        if (number % factor == 0)
        {
            factors.Add(factor);
            if (factor != number / factor)
            {
                // Don't add the square root twice!
                factors.Add(number / factor);
            }
        }
    }

    string f = string.Join(",", factors.ToArray());
    string s = "random num is " + number +
              ", and the factors of this number are " + f;
    return s;

}
```

```
    public string bigger(string a)
    {
        string s;
        char[] chars = a.ToCharArray();
        int num1 = (int)Char.GetNumericValue(chars[0]);
        int num2 = (int)Char.GetNumericValue(chars[chars.Length - 1]);
        if (num1 - num2 > 0)
        {
            s = a + " is true";
        }
        else
        {
            s = a + " is false";
        }

        return s;

    }




    public string smaller(string a)
    {
        string s;
        char[] chars = a.ToCharArray();
        int num1 = (int)Char.GetNumericValue(chars[0]);
        int num2 = (int)Char.GetNumericValue(chars[chars.Length - 1]);
        if (num1 - num2 < 0)
        {
            s = a + " is true";
        }
        else
        {
            s = a + " is false";
        }
        return s;

    }

}
```

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