ウルトラチャレンジ問題の説明

24115172 Yoon SeungYong

2012. 07. 24.

概 要

この文書は平成２４年度プログラミングI講義の「ウルトラチャレンジ問題２０１２」を解き、その開発過程かつ補足説明を述べるものである。特にこの文書は問題４のウエブブラウザの開発過程を詳細に扱っている。

HTML2.0の規格を遵守している問題４のウエブブラウザ「YBrowser」はWindows 7日本語版でMicrosoft Visual C# 2010を用いて.NET Framework 4.0を基盤として開発し、githubの分散バージョン管理システムで管理されている。

目 次

第1章　ウエブブラウザ

ウルトラチャレンジ問題２０１２
問題４。ウエブブラウザを作成せよ。

1.1　問題分析

1.1.1　規格の採用

ウエブブラウザはHyperText Markup Language(HTML)で作成された文書を解釈し表示するプログラムである。特にHypertext Transfer Protocol(HTTP)を通じてインターネット上のHTML文書を表示するのが主な目的である。現在HTML4.01またXHTML1.0の規格が発表されウエブサイトに通用されていて、次世代の言語であるHTML5が標準化の手順を踏んでいる。HTMLを視覚化するウエブブラウザではMicrosoft Internet Explorer, Mozilla Firefox, Google Chromeなどが主に使われている。

ウルトラチャレンジ問題の解答としてウエブブラウザとして最新標準であるHTML4.01を具現したら最高であろうが、２週間の短い期間ではHTML4.01の数百ページのRFC文書を読むことすら容易くないので設計の範囲を縮小するしかない。よって旧規格であるHTML3.0とHTML2.0に従い開発することにした。ところがHTML3.0とHTML2.0は1997年HTML4.0の発表と同時に廃止されたので実際作らない方がよいとされる。今回の解答としてはHTML2.0の規格を守るウエブページを表示するウエブブラウザの提出を目標とした。

1.1.2　HTML 2.0の分析

HTMLはテキスト情報処理のISO標準であるStandard Generalized Markup Language(SGML)を利用して定義された言語である。よってHTML

の各要素を SGML で定義した Document Type Definition(DTD) が存在し、DTD を基準として HTML を分析すればよい。

RFC1866 で定義している HTML 文書は SGML で作成されて HTML DTD を遵守する RFC1866 の慣習を守るものである。HTML DTD に定義された要素中 HTML.Recommended は有用に使えるが必ず要るものではないものに付けられ、HTML.Deprecated は旧バージョンで使われたが何らかの理由で使わないことを進めているものに表記されている。User Agent つまりブラウザは次に従うべきだ。HTML 文書を SGML に従いデータ列とマークアップ要素で常に同じく分解できリンクを通じて文書間の移動が可能である。第２水準ブラウザは追加的にフォームフィールドを用いて情報の交換が可能なものである。

HTML 文書での要素たちは木の構造をしている。一部の要素は始めのタグと終わりのタグがあり、またタグの間で互い影響を与える場合もある。

HTML の要素は大きくデータ列、タグ、アトリビュート、コメントとして分けられる。データ列は画面に表示されるテキストを表す。タグに用いられるが打ちにくい文字は特集な方法で表す。< のように文字の名前を入れたり、< のごとく文字コードを用いて表示するものがある。ただし、; は場合により省略できるが文字の間に空白を入れるのは許さない。始めのタグは常に¡で始まり¿で終わらなくてはならない。同じく終わりのタグは常に¡/で始まり¿で終わる。¡また¡/の後にタグの名前が入り、その間の空白は許さない。ところが¿の前の空白はあり得る。タグ、アトリビュート、特殊文字などに使われる名前は文字、数字、点、ハイフンで 72 字以内で構成され大文字と小文字の区分をしない。ただし、特殊文字の名前は例外として大文字と小文字の区分をする。アトリビュートは始めのタグにあるものでアトリビュート名と等号、値で構成されるがアトリビュート名だけで構成される場合もある。また等号の周りの空白は許容される。文字列の値の場合ダブルクオーテーションやシングルクオーテーションで囲まれるがその間に同じクオーテーションがあってはならない。コメントは¡!で始まるタグで、複数の行を含む場合は¡!–で始め–¿で終わらせる。トークンを分けるとき消した方がよいとされる。

HTTP を用いて通信する場合サーバは転送するものが HTML 文書であることを Media Type を text/html で表記して示す。

もし分析途中未確認の要素があったらタグやアトリビュートの場合無視して、特殊もにはそのまま残すことを原則とする。

1.2 プログラム設計

1.2.1 ライブラリ

まず、ウルトラチャレンジ問題らしく.NET Framework で提供している WebBrowser コントロールと HtmlDocument などの要素を使わないことにした。

HTML 文書を表示するため二つのことを考えた。

1. RichTextBox を用いる
2. GDI+ツールで絵として直接描く。

1.2.2 構造

- Core：HTML の元を宣言
  - HTML の各要素を保存するクラス。
    - Attribute
    - AttributeCollection
    - Element
    - Document
  - HTML の分析に関わるクラス。
    - HtmlTokenizer
    - HtmlParser
    - HtmlReader
- Test：各コントロールをテスト
- YWebView：画面表示コントロール
  - YWebView：UI 担当
  - NetProcess：ネットワーク担当
- YBrowser：Win32 アプリケーション

1.3 プログラム開発

全ての開発過程は github に保存されている。

```
https://github.com/forcom/y-browser/
git-clone: git@github.com:forcom/y-browser.git
```

開発順番は全体的にプログラムの構造の順番と同じである。

1. Core.HTML の要素
2. Core.HtmlTokenizer
3. Core.HtmlParser
4. Core.HtmlReader
5. Test
6. YWebView.NetProcess
7. Core.HtmlTag
8. YWebView.DrawPage
9. YWebView.YWebView
10. YBrowser

1.3.1 Core.HTML の要素

最初にこの段階で訂正のコミットが相当多いことが分かる。一番重要な部分であるので数回のテストを通して適切な構造を作った。

1.3.2 Core.HtmlTokenizer

HTML の各要素を分離するクラスである。この部分は予想より早く終わった。去年 C#で Lisp を具現した経験が役に立った。正規表現式を用いてトークンを分離して、DTD で定義されたタグとアトリビュートを前処理したマップにトークンをあわせて簡単に済ませた。

1.3.3 Core.HtmlParser

分離したトークンがHTML DTDの定義通り正確に構成されたかを確認する部分である。この段階でコメントと定義されていないタグは無視して削除した。またツリー構造をなっていないタグも適切に調整するようにした。

1.3.4 Core.HtmlReader

HtmlTokenizerとHtmlParserをカプセル化するクラスである。ここでHTML文字列をDocumentクラスで得られる。

1.3.5 Test

HTML2.0を遵守するページをテストケースとしてCoreをテストした。

1.3.6 YWebView.NetProcess

HTTP通信を担当するクラスである。YWebViewで処理するか分離するか悩んでいたが分離したのが正解であった。ネットワークプログラミングはソケットプログラミングの代わりにSystem.Netを用いて作業した。ソケットコーディングを始めるとHTTPに関するRFC文書を見ることになるからである。

1.3.7 Core.HtmlTag

DrawPageの作業を始める際にタグが少し扱い難くてHtmlTokenizerとHtmlParser を見直すようになった。HTML DTDの前処理が結局同じなのにHtmlTokenizerとHtmlParserが独自に作っていることを発見し、HTML DTDの宣言のため作られた各種マップをHtmlTagに集めて一つにあわせた。他のタグの作業や定義ができるようにする目的もある。その例がHtmlTag.ListTypeである。

1.3.8 YWebView.DrawPage

DrawPageを作業する前にRichTextBoxを使う計画を持っていたがRichTextBoxがMicrosoftが規定しているRTFの水準まで及んでいないことを気づき、直接絵を描いて表示することを選んだ。絵を描くことの一番難しいところはテキストを描くことである。画面の右端に達したら区切ってまた左から始めないとならない。GDI+ツールのMeasureStringを用いて丁寧に区切りを探して表示させた。計画を急に変え締め切りに迫られ多少気早に作業した感じがあってコメントも付けず、コードが少し汚くなった感じがする。また基準点を描く対象の左上に設定したが作業が殆ど終わった頃左下の方が綺麗に描けることに気づいたが締め切りにあわせない感じがして現状を維持することにした。

1.3.9 YWebView.YWebView

DrawPageの作業と同時にYWebViewのところも並行した。元々YWebViewがDrawPageやNetProcessより上位のコントロールであるので当然のことだろう。YBrowserに貼り付ける準備のためイベントなどを実装した。

1.3.10 YBrowser

大部分の機能がYWebViewに実装されていたのでアドレスバーとステータスバーを挿入しYWebViewを貼り付けることでYBrowserが完成になった。

1.4 テスト及びデバッグ

テストとデバッグは開発の間でユニットテスト形式で行われた。完成度テストは次のように直接ウエブページを航海した画面と既存のウエブブラウザの画面の違いで判断した。

1.4.1 HTML 2.0 ページ

```
Test Site : http://www.w3.org/MarkUp/html-spec/
            ここで繋がっている HTML2.0 のページ。
```

殆ど正確表示されているがイメージの位置やホワイトスペースの間隔、リストの表示が少し不安定であるところが見える。ページの移動は順調に行っているが時々理由不明でアクセスができないところがある。またブックマークには移動ができない。でもウエブページとして読めないぐらいではない。

1.4.2 上位バージョンのページ

```
Test Site : http://www.google.com/
```

当たり前のことであろうが正確に表示されていない。

1.5 自己評価

短い期間であったがウエブブラウザが作られたのは嬉しいことだ。ところが完璧に HTML2.0 を具現することにはまだ遠く見える。Internet Explorer などの常用プログラムでは GPU の加速技術を使うなど最適化されている。このような技術を学ばないといけないと思う。また、何気なく使っているコントロールが実際には作りにくいものであることを実感する機会になった。

第2章　開発マニュアル

この章ではYBrowserの開発のための環境且つビルドの手順を述べる。

2.1　コンパイル

2.1.1　ソースコードの取得

このプロジェクトはgithubで公開プロジェクトとして管理されているのでgitを用いて最新のソースコードを取得することができる。次の命令を通じて簡単に入手可能である。

- `git clone git@github.com:forcom/y-browser.git`

2.1.2　システム環境

このプログラムはWindowsシステムを基盤として開発されたので次のような環境を求められる。

- Windows Vista 以上
- .NET Framework 4.0
- Microsoft Visual Studio 2010 (C#付き)

2.1.3　ビルド手順

このソリューションは次の順序でプロジェクトをビルドし、バイナリが得られる。

1. Core
2. YWebView
3. YBrowser

第3章　操作マニュアル

この章ではYBrowserの操作方法を述べる。

3.1　実行方法

.NET Framework 4.0 再配布パッケージが取り付けられたWindows Vista以上日本語版のパソコンでビルドされたYBrowser.exeを実行する。ただし、YBrowser.exeの同じ経路にCore.dll, YWebView.dllがあることを確認すること。

3.2　操作法

プログラムの画面は上端にアドレスバーと移動ボタンがあり、真ん中にビューアが配置され、下端にはステータスバーが位置している。

アドレスバーに移動しようとしているウエブサイトのURLを入力しEnterキーを押すか移動ボタンをクリックする。ページ移動中にはステータスバーに'Opening Page...'と表示され、読み込みが終わったとき'Completed.'と表す。

ビューアでリンクにカーソルをかざすと手の模様で変わる。リンクをクリックするとそのリンクのページに移動する。また説明のあるイメージやリンクにカーソルをかざすとその説明がツールティップで現れる。

関連図書

[1] IETF, RFC 1866: Hypertext Markup Language - 2.0, 1995.

[2] Harold Abelson 外, Structure and Interpretation of Computer Programs 2/E, MIT Press, 1996.

[3] Microsoft, Microsoft Developer Network Library, 2012.