HTML 校正ソフトウェアの開発

06H008 上森 史利

目次

概要

HTML 4.01 が策定されて 10 年が経つ。HTML 4.x は以前のバージョンとは違い、デザインは CSS を用いて、HTML は文書の論理構造のみを記述する。これにより、メンテナンス性の向上、検索エンジン最適化、アクセシビリティの向上がはかれる。これはウェブサイトを制作する側、利用する側双方に対してメリットである。しかし、多くのウェブサイトは HTML 4.01 の仕様に沿っていない。その理由として既にコンテンツとして完成した HTML を HTML 4.01 の仕様に変更するための時間や知識が、ユーザーに無いからではないかと私は考えた。この問題を解決するためには、知識の無いユーザーでも簡単に仕様に沿った HTML 文書に変更できるソフトウェアが必要である。そこで私は変更を加えたい HTML ファイルを指定するだけで、仕様に沿った HTML に校正してくれるソフトウェアを開発した。動作の流れとして、ユーザー側でソフトウェアに校正するファイルを指定する。次に保存したいディレクトリの選択とファイル名を入力し、校正を開始するボタンを押すだけである。なお、本ソフトウェアは保存ファイル名の入力以外の操作は全てマウスから行うことができる。ソフトウェアは受け取ったファイル内を走査する。本ソフトウェアで処理する項目は大きく分けて 3 つある。1 つ目に、HTML 4.01 で非推奨とされている要素の削除。2 つ目に、ブロック要素に内包できない要素の削除。3 つ目に、テーブル要素の文法の比較と削除。最後に、校正した HTML ファイルをユーザーが指定したファイル名で保存する。本ソフトウェアをテストした結果として、本研究で開発したソフトウェアにより一部の HTML 文法の校正に成功した。また時間や知識の無いユーザーの手間を減らすことにも成功した。現状の問題点として仕様に沿わない要素を削除するのみで追加はしないため、元の HTML ファイルに要素が足りない場合に対処ができない。また、属性の理解と判断ができない。今後の課題として、現状の問題と併せてソフトウェアのユーザーインタフェイスの改良が必要である。

1 はじめに

HTML 4.01 が 1999 年に策定されて今年で 10 年となる。HTML 4.x は以前のバージョンとは違い、HTML には文書の論理構造のみを記入し、デザインは記述しない。これにより、メンテナンス性の向上。アクセシビリティの向上。検索エンジン最適化などのウェブページを制作する側、利用する側双方に対してインターネットがより有用なものになる。しかし、未だに W3C[*1][1, 2] の定めた仕様に沿っていないウェブサイトが多数存在している。その一因として既に仕様に沿っていない HTML 文法で書かれたウェブページを持っている製作者が、自分のページを仕様に沿った HTML に書き換えていないことが問題ではないかと考える。または、製作者本人に既存のウェブページを書き換える意識があったとしても、HTML に対する知識が無い場合やまたはその逆で HTML に関する知識を持ちながら、処理に掛かる手間や時間等の理由から、改善する意識が意識が無い場合もありえる。しかし、HTML を学ぶことや既存のウェブページを書き換えることは、製作者の時間を大きく奪うものである。そのような問題を根本的に回避するには、製作者がウェブページの変更に時間を割かれずにウェブページを正しく書き直せるソフトウェアが必要である。そこで今回、既存のウェブページを HTML 4.01 の仕様に沿ったものに変換するソフトウェアを開発し、その効果を検証した。論文の構成として、論文の骨組みでこの論文内で使われる用語や HTML の基礎的な説明とシステムの概要。各処理のイメージ。結果と考察で実際の処理結果をまとめる。まとめで今回のソフトウェアで有用性と今後の課題を示す。

[*1] WWW で利用される技術の標準化をすすめる団体。WWW 技術に関わりの深い企業、大学・研究所、個人などが集まって、1994 年 10 月に発足した。

2 用語とソフトウェアの説明

この章では HTML に関わる用語の定義と基礎知識を記述した後、ソフトウェアの各処理を解説をおこなう。

2.1 HTML と HTML 文書

HTML とは、HyperText Markup Language を省略した言葉で、W3C が策定したマークアップ[*2]言語である。最新版は 1999 年 12 月に策定された HTML 4.01。その仕様に沿って書かれた文書を HTML 文書と呼ぶ。文書中にタグと呼ばれる文字で文を指定することでブラウザ上で文書を閲覧や画像や音声、動画、特殊な効果や他の HTML 文書へのリンクを埋め込むことができる。HTML 文章とは仕様書 (DTD) に定められた記述方法に従った文書。タグ (TAG) によって要素 (ELEMENT) を宣言し、マークアップすることでテキストなどの情報に構造上の意味を与えた文書。

2.2 W3C

W3C とは、World Wide Web Consortium の略で、WWW 技術に関わりの深い企業、大学・研究所、個人が参加している国際的な産学官共同コンソーシアム。発足は 1994 年 10 月。活動内容は Web で利用される技術仕様の標準化。技術仕様は草案、勧告候補、勧告案を経て、最終的に勧告が行われるが、各段階で仕様を公開し、一般のユーザーから広く意見を求める。その意見の取り纏めと次回仕様への反映を行う。Web の持つ機能を最大限に引き出すと同時に、Web のユニバーサルアクセス[*3]を実現するのが目的。

2.3 要素

製作者が意図的に他と区別しようとした文書などの情報。HTML ではタグによって宣言される。全ての要素には親子関係がある。

2.3.1 ブロック要素

ブロック要素とは HTML 文書構造を構成する基本要素。HTML 4.01 では (例外を除き) ブロック要素内でのみ文書を記入してよい。要素によって内包できる要素が違う。

2.3.2 インライン要素

インライン要素とは文章に特定の機能や役割を持たせる要素。いずれの要素もインライン要素を内包することはできるが、ブロック要素を内包することは認められていない。

[*2] マークアップとは HTML でのマークアップとは、文書に効果を与えるために (多くの場合) タグで文書の前後を囲い目的の要素を宣言すること。

[*3] 「言語、文化、あらゆる機器からのアクセス、障害の有無に関わらず」と言う意味。

2.4 HTML 4.01

HTML 4.01 は、HTML 4.0 を全般的に改訂したもの。HTML とは本来、文章の論理構造を記述する言語であった。しかし以前の規格である HTML3.2 から、見た目に関する要素拡張の結果、見栄えを記述するタグが取り込まれた。そこで、HTML 4.01 では文書の論理構造を記述するという本来の目的に立ち返り、フォント情報、配列、色のような文書のレイアウトに関する見栄えの部分は CSS[*4]で設定する。現在の標準的な HTML の仕様である。以前のバージョンの HTML との互換性を保つために 3 種類の DTD を持っている。

2.5 DTD

Document Type Definition の略で「文書型定義」の意味。文書を記述する際、その文書中でどのような要素や属性が使われているかを定義したもの。Strict,Transitional,Frameset の 3 種類があり、それぞれ使用が許されている要素と属性に違いがある。今回の研究で制作するソフトウェアでは 3 種類の DTD の中で、もっとも厳密に定義されている DTD である Strict を基準とする。フレーム、見た目に関する要素と属性は原則非推奨または禁止。以下の記述は Strict の DTD 宣言である。

<DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN""http://www.w3.org/TR/html4/strict.dtd">

2.6 Another HTML-lint との違い

Another HTML-lint [3] と今回開発したソフトウェアとの違いを**表 1** に示す。

表 1　Another HTML-lint との比較

	Another HTML lint	開発した HTML 校正ソフトウェア
チェック方法	DTD に則した文法 アクセシビリティガイドラインに準拠した項目 作者が望ましくないとした項目	DTD(Strict) に則した文法
チェック後の処理	警告文 警告数に応じた点数表示	違反要素の削除
利用方法	URL を指定する 入力欄に直接ソースコードを記述する ローカルファイルをアップロードする	ソフトウェアで HTML ファイルを指定する

今回開発したソフトウェアと Another HTML-lint とのもっとも大きな違いは、文法的ミスを発見した場合の処理方法にある。Another HTML-lint は文法的ミスを発見した場合の処理として警告を行うが、実際の HTML 文書に変更を加えるのはユーザーであった。しかし、本ソフトウェアはユーザーの手間、時間等の観点から、文法的ミスがあった場合、自動で HTML 文書に変更を加える。

[*4] Cascading Style Sheet の略で Web ページのレイアウトを定義する規格。

2.7 システムの概要

システムの概略図を**図 1** に示す。ユーザーはソフトウェアのインタフェイスに「(校正する)HTML ファイルのパス」、「保存先ディレクトリ」、「保存ファイル名」の 3 つの情報を入力する。ソフトウェアはユーザーから受け取った各情報に従い、HTML ファイルの読み込み、校正、HTML ファイルの出力を行う。

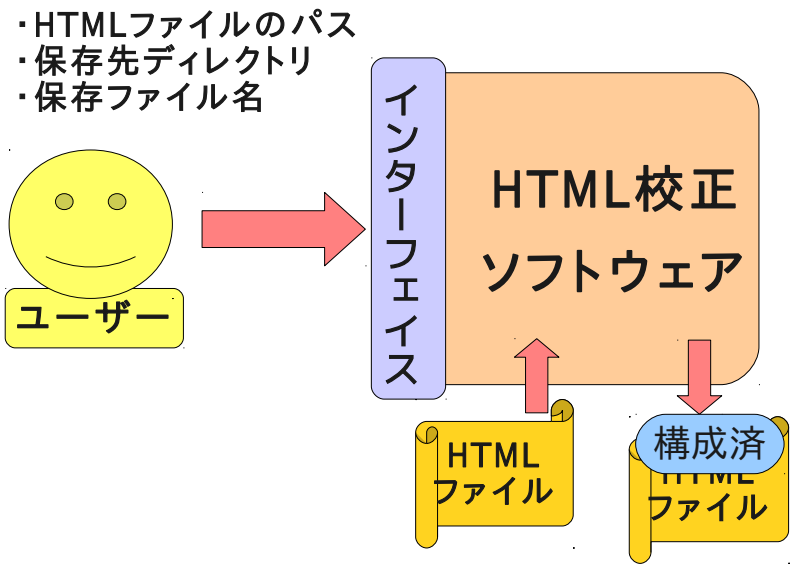


図 1　システムの概略図

2.8 開発環境

- ハードウェア (IBM-PC/AT 互換機)
  - CPU:Intel(R) Celeron(R) CPU 2.80GHz
  - Memory:256KB
- OS：Vine Linux 4.2　/ Ubuntu Linux 9.04　[研究室内マシン/ 個人所有マシン]

- 開発言語：python[4] 2.5 / 2.6 [研究室内マシン/ 個人所有マシン]

2.9 処理の流れ

本ソフトウェアはすべて GUI で操作可能である。実際のインタフェイスを**図 2** に示す。

1. 「校正する HTML ファイル」をクリックして HTML ファイルを選択
2. 「保存先選択」をクリックして校正した HTML ファイルを置きたいディレクトリを選択
3. 校正した HTML ファイルの名前をフォームに入力
4. フォームに入力したファイル名[*5]を「出力ファイル名確定」で確定する
5. 以上の条件で問題なければ「実行」ボタンをクリックすると HTML ファイルの校正が開始される

[*5] インタフェイスから入力された文字列に.html を追加したものをファイル名として出力される。

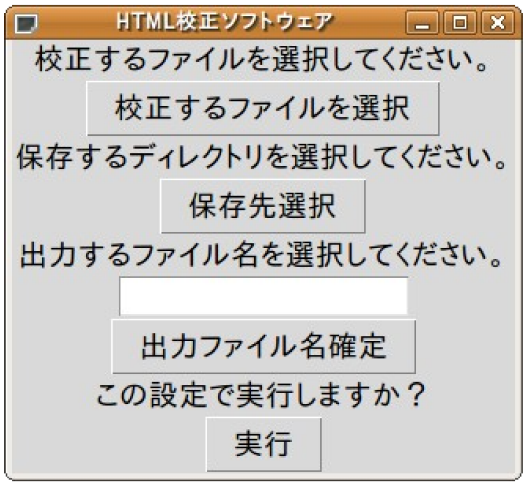


図 2　入力インタフェイス

また、ターミナル上でもコマンドライン引数として、「校正するファイル」、「保存先ディレクトリ」、「出力ファイル名」の順に入力することでも校正を行うことができる。その実行例を**図 3** に示す。

図 3　ターミナルからの実行

2.10　各処理の説明

2.10.1　非推奨要素の削除

本ソフトウェアでは非推奨要素[*6]があった場合、マークアップされている文書を残して要素のみを削除する。
削除する条件と処理

- 本文中に非推奨要素があった場合
  - 該当する要素を削除する

上記の一覧に基づき、処理のイメージを**図 4**、非推奨とされた要素を**表 2** にそれぞれ示す。

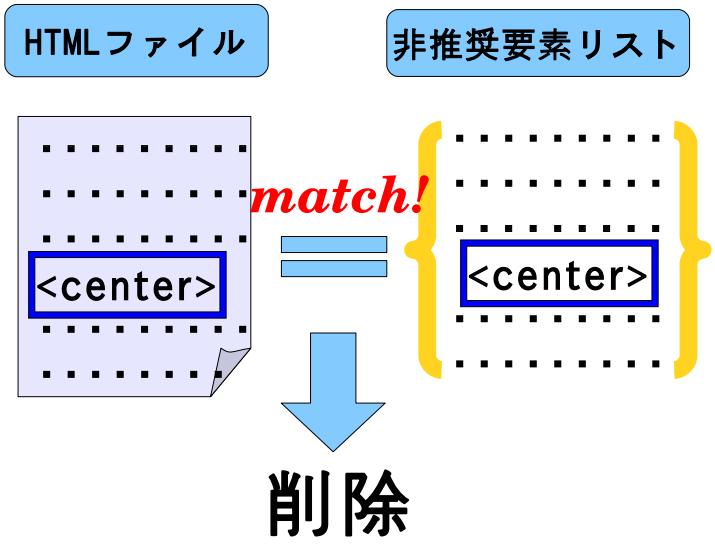


図 4　非推奨要素処理の概略図

<applet>	<basefont>	
<dir>	<font>	<isindex>
<listing>	<menu>	<plaintext>
<s>	<strike>	<u>
<xmp>		

表 2　非推奨要素一覧

[*6] 非推奨要素とは以前の HTML のバージョンでは使われていたが、HTML 4.01 に変更された際に何らかの問題があり、使うことが推奨されなくなった要素のこと。

2.10.2 ブロック要素の親子関係正常化

ブロック要素には各要素ごとに自分の子として持てる要素が定められている。ブロック要素の子要素を HTML 内の要素と比較し、認められていない要素が含まれていた場合は削除する。

削除する条件と処理

- 各ブロック要素の内部に内包できない要素があった場合
  - 該当する要素を削除する

上記の条件を元に [*7]を例として処理のイメージを**図 5** に示す。

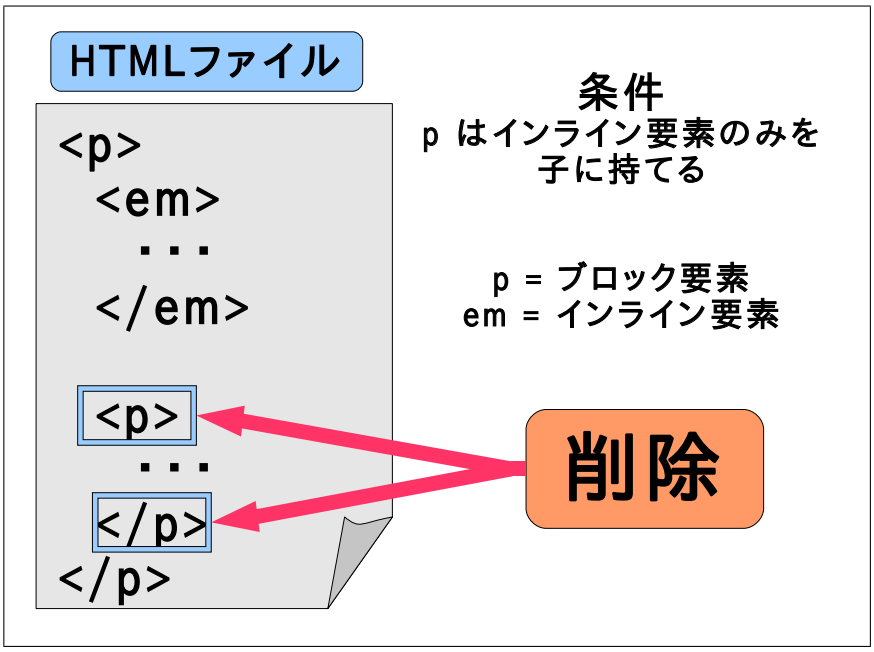


図 5　ブロック要素の処理イメージ

図 5 の場合、 の子要素に、 と、インライン要素である[*8]の 2 つの要素を持っている。しかしブロック要素である は自分の子要素としてインライン要素しか持つことが許されていない。よって本ソフトウェアは内部にある 要素を削除する。

各ブロック要素が子として持てる要素のパターンは大きく分けて 4 種類に分けられる。それらのパターンと要素の内訳を以下にまとめた。(非推奨または禁止の要素は除外。)

- 自分を含む全てのブロック要素
  - blockquote・・・引用文であることを示す。
  - noscript・・・スクリプトが動作しない環境での代替コンテンツであることを示す。
- 全てのインライン要素
  - address・・・著者の情報・連絡方法を示す。
  - p・・・段落を構成する。
  - hn・・・文章の見出しを示す。
- 特定のブロック要素またはインライン要素

[*7] Paragraph の略で、 ～ で囲まれた部分がひとつの段落であることを表す。
[*8] Emphasis の略で、～で囲まれた部分が強調される。

  - dl･･･定義のリストあることを示す。
  - fieldset･･･フォームコントロールのグループを示す。
  - ol･･･順序付きリストを示す。
  - pre･･･整形済みテキストの節であることを示す。
  - ul･･･順序なしリストを示す。
  - table･･･表を構成する要素の大枠を示す。
- 全てのブロック要素とインライン要素
  - div･･･ブロックレベルでのスタイルコンテナ。
  - noframes･･･フレーム表示されないときのコメントを示す。

上記の「特定のブロック要素またはインライン要素」が内包できる要素を以下にまとめる。

- dl
  - dt･･･定義する用語であることを示す。
  - dd･･･定義の説明であることを示す。
- fieldset
  - ブロック要素
  - インライン要素
  - legend･･･フォームの入力項目グループにタイトルを付ける。
- ol
  - li･･･リストの項目を示す。
- pre
  - img･･･埋め込み画像であることを示す。
  - object･･･さまざまな形式のデータを配置。
  - applet･･･Java アプレットを挿入する。
  - big･･･文字を大きくする。
  - small･･･文字を小さくする。
  - sub･･･文字を上付きにする。
  - sup･･･文字を下付きにする。
  - font･･･フォントの種類、大きさ、色を指定する。
  - basefont･･･テキストの基準サイズ、基準色、基準フォントを指定する。
  - 上記を除くインライン要素。
- ul
  - li･･･リストの項目を示す。
- table
  - caption･･･表題を示す。
  - colgroup･･･構造的な列のグループを示す。
  - col･･･列のグループを示す。
  - thead･･･表のヘッダを示す。
  - tfoot･･･表のフッタを示す。

2.10.3　正しい要素を満たしていないテーブルの削除

ブロック要素の中でもテーブル要素については特殊で、子として持てる要素が特別に制限されている。しかし、現段階では文法に違反した要素は削除されるため、文法定義を厳密にした場合多くのテーブルが削除されてしまい、HTML 文書としての体裁を崩す恐れがある。よってテーブル要素のみ、例外として一般に使用すると思われる要素のみを比較対象とした。

削除する条件として、<table>直下の子要素として<tr>以外があった場合 、<tr>直下の子要素として<td>以外があった場合 。もしいずれかの条件に当てはまった場合、本文を除くテーブル要素以下の要素を削除する

処理のイメージを**図 6** に示す。

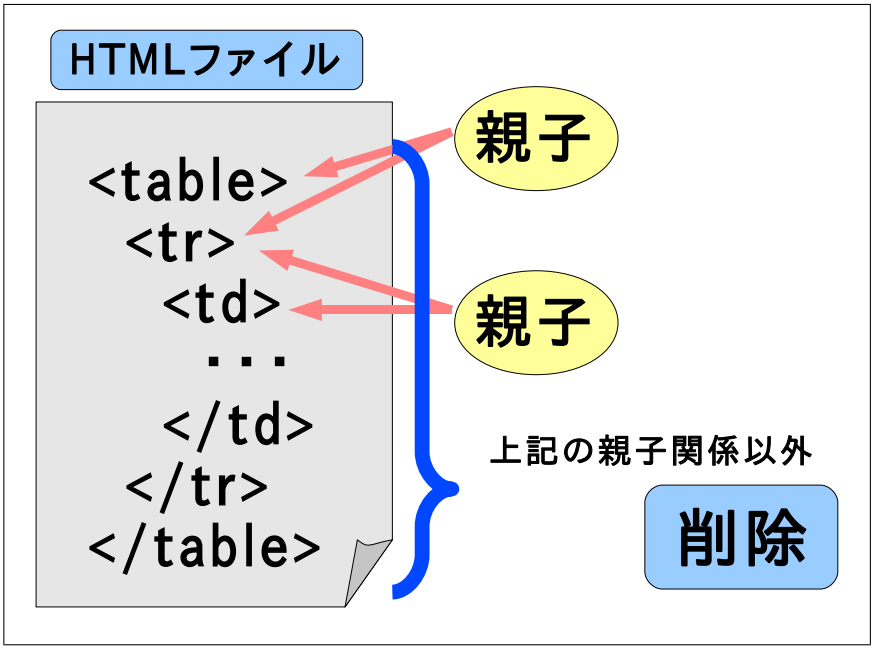


図 6　テーブル要素処理の概略図

今回、簡略化して解釈したテーブル要素の文法は、<table>を親として、行を示す<tr>のみを子として持つことが許される。そして<tr>の子としてもてるものは<td>のみとした。また、文書を記述できるのは<td>の内部だけとした。上記の条件に当てはまらないテーブルを発見した場合、本文を除くテーブル用タグ自体を削除する。

3　結果と考察

この章では 2.10 章「各要素の実際の処理」で説明した条件を用いて、実際に HTML ファイルを校正した結果を示す。HTML ファイルのソースコードの一部と、ブラウザでの実際の表示の 2 種類を処理を行う前後で 2 パターンずつ掲載する。使用するブラウザは FireFox(バージョン 3.5.5) を用いた。

3.1　各要素の実際の処理結果

3.1.1　非推奨要素の削除

実際にテストを行い処理前と処理後に分けて、ブラウザで表示したものを**図 7** と**図 8** に、ソースの処理前と処理後をそれぞれ**図 9** と**図 10** に示す。

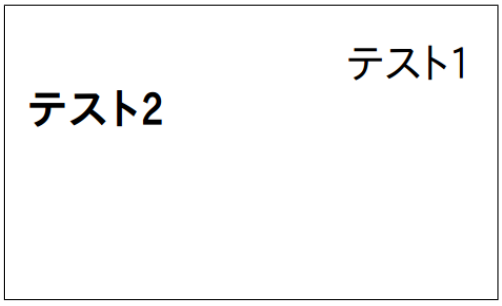


図 7　処理を行う前のブラウザでの表示

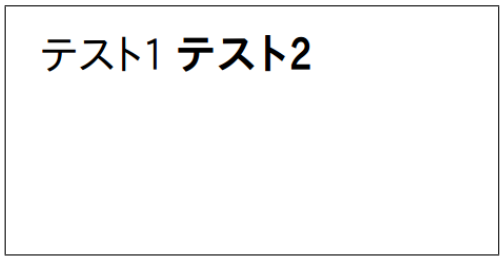


図 8　処理を行った後のブラウザでの表示

```
<html>
<head>
<title>test.html</title>
</head>
<body>

<center>
テスト1
</center>


<strong>
テスト2
</strong>

</body>
</html>
```

図 9　非推奨要素処理前のソース

```
<html>
<head>
<title>test.html</title>
</head>
<body>


テスト1


<strong>
テスト2
</strong>


</body>
</html>
```

図 10　非推奨要素処理後のソース

図 7 と**図 8** を比べると「テスト 1」と「テスト 2」は同じ行に移動している。

処理前のソースである**図 9** と処理後のソースの**図 10** を比べると「テスト 1」がマークアップされていた が削除されている事が分かる。これは **2.10.1** 章の「非推奨要素の削除」に定めた条件に基づき、非推

奨要素である 要素のみが削除された。

3.1.2 ブロック要素の親子関係正常化

実際にテストを行いブラウザで表示させたものを**図 11** と**図 12** に示す。また、処理を行う前と後のソースをそれぞれ**図 13** と**図 14** に示す。

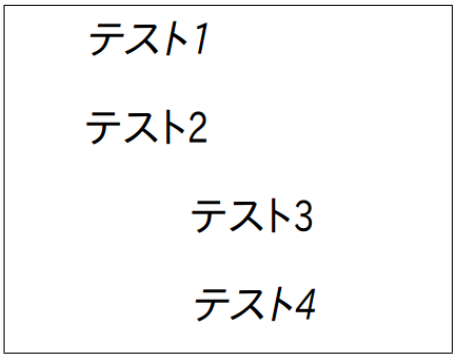


図 11　処理を行う前のブラウザでの表示

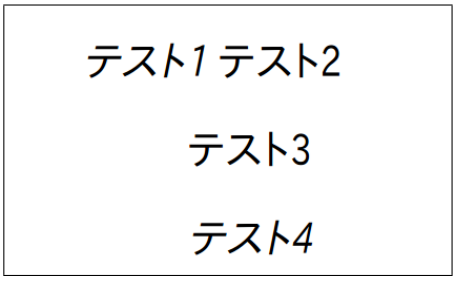


図 12　処理を行った後のブラウザでの表示

```
<html>
<head>
<title>test.html</title>
</head>
<body>
<p>

<em>
テスト1
</em>
<p>
テスト2
</p>

</p>
<blockquote>
<p>
テスト3
</p>
<em>
テスト4
</em>
</blockquote>
</body>
</html>
```

図 13　ブロック要素の処理前のソース

```
<html>
<head>
<title>test.html</title>
</head>
<body>

<p>

<em>
テスト1
</em>

テスト2


</p>
<blockquote>
<p>
テスト3
</p>
<em>
テスト4
</em>
</blockquote>
</body>
</html>
```

図 14　ブロック要素の処理後のソース

処理前の**図 11** と、処理後の**図 12** を比べると、「テスト 1」と「テスト 2」が同じ行に存在する事が分かる。これは **2.10.2** 章「ブロック要素の親子関係正常化」に定めた条件に基づき、「テスト 1」、「テスト 2」が含まれる<p>が文法的にブロック要素を子として持つことができないため、内包された<p>要素が削除された。また、<blockquote>はブロック要素、インライン要素ともに、子として持てるため内包した要素はいずれもも削除されなかった。

3.1.3 正しい要素を満たしていないテーブルの削除

実際にテストを行いブラウザで表示したものを**図 15** と**図 16** に示す。なお、ブラウザ上での違いを分かり易くするため、border 属性[*9]の値を 1 とした。また、処理を行う前と処理を行った後のソースを、それぞれ**図 17** と**図 18** に示す。

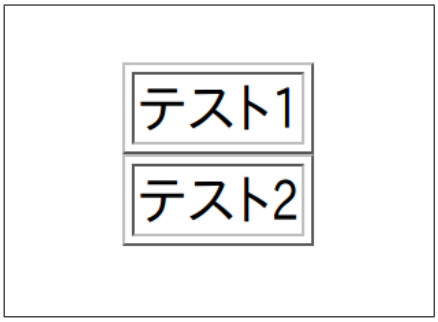


図 15　処理を行う前のブラウザでの表示

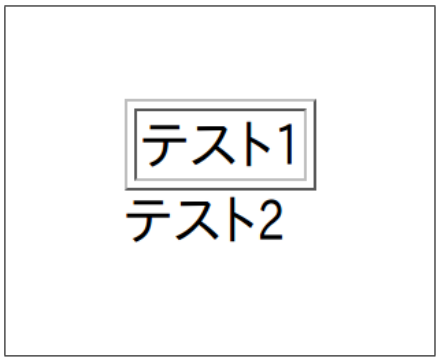


図 16　処理を行った前のブラウザでの表示

処理前の**図 15** と、処理後の**図 16** を比べると、「テスト 2」が含まれるテーブルのみが削除された事が分かる。これは **2.10.3** 章「正しい要素を満たしていないテーブルの削除」に定めた条件に基づき、「テスト 1」の含まれているテーブルは削除されなかった。一方、「テスト 2」の含まれるテーブルは</td>が無いため「テスト 2」を除くテーブル全体が削除された。

3.2 考察

実例の結果から、ソフトウェアの利点と問題点を考察する。

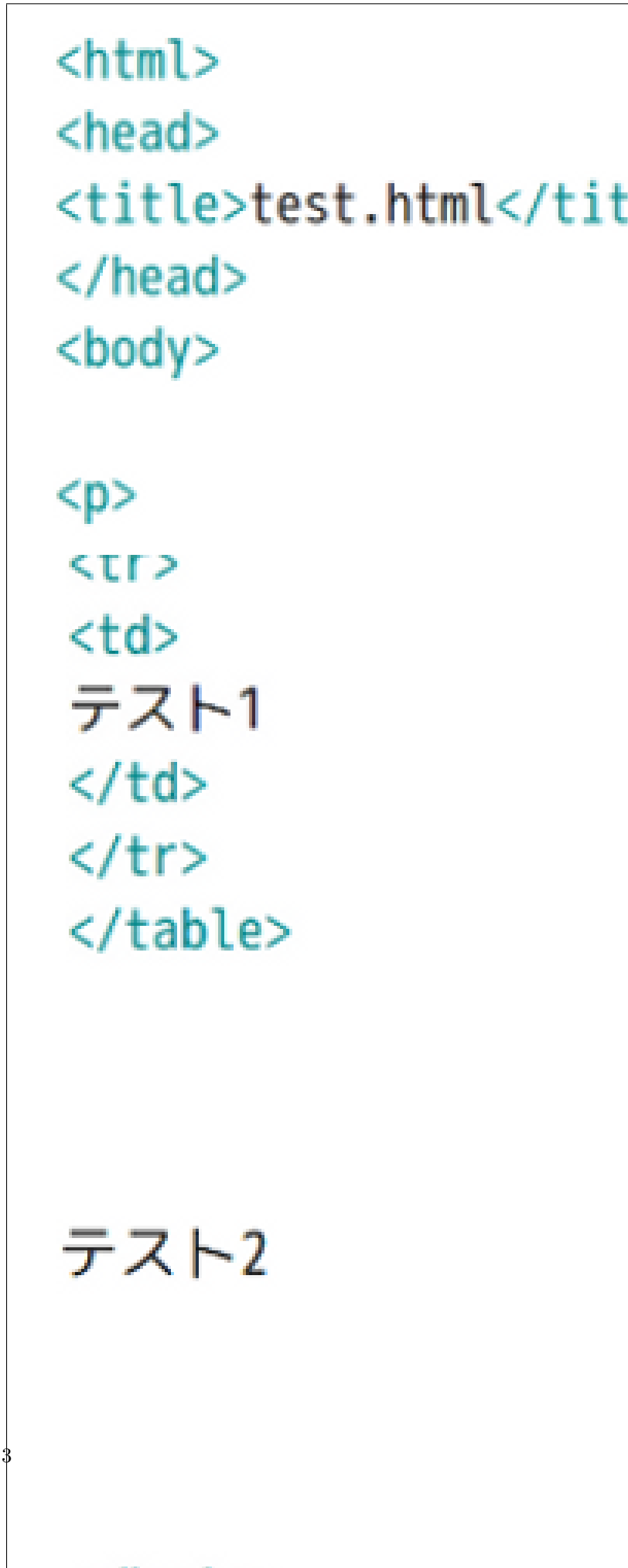
3.2.1 利点

- HTML を正しく校正できる
  - 今回、実装した「非推奨要素の削除」、「ブロック要素の親子関係正常化」、「正しい文法を満たしていないテーブル要素の削除」の 3 点において HTML の校正に成功した。
- 作業工程と作業時間の短縮
  - ユーザーが手作業でおこなう校正に比べて、作業工程を減らすことに成功した。また、相対的に作業時間の短縮にも成功した。
- ユーザーに HTML の知識が無くても使用できる
  - 表示されている指示の通りに操作するだけで校正が可能。また、校正に必要な操作は「保存ファイル名」の入力を除いて、全てマウスからおこなえる。

[*9] table 要素に使用できる要素のひとつ。テーブルの枠組みの太さを指定できる。

```
<html>
<head>
<title>test.html</title>
</head>
<body>

<table border="1">
<tr>
<td>
テスト1
</td>
</tr>
</table>

<table border="1">
<tr>
<td>
テスト2

</tr>
</table>

</body>
</html>
```

図 17　テーブル要素の処理前のソース



3.2.2 **問題点**

- 削除だけで、足りない要素の追加ができないので場合によってユーザーの手間が増える
  - 対象の HTML ファイルに加える変化は削除のみである。よって、要素が足りない場合の文法ミスを校正することができない。
- 属性の判断ができない
  - 各要素に付加される属性のうち、使用が推奨されていない属性がある。今回開発したソフトウェアでは非推奨要素の校正は行うが、非推奨属性の校正は行えない。
- GUI のみで操作する場合、変換するファイルをひとつしか指定できない
  - ウェブサイトは通常、複数のウェブページが集まり構成されている。しかし、本ソフトウェアを GUI で操作する場合、一連の操作 1 回に付き、1 つの HTML ファイルしか校正することができない。

4 まとめ

本研究では文法的に間違っている HTML ファイルを自動で校正するソフトウェアを開発した。これにより、HTML の知識が無いユーザーや HTML を手動で校正するリソースの無いユーザーにも手軽に早く校正した HTML ファイルを提供することができる。テストの結果、非推奨または廃止された要素の削除とブロック要素の内包できない要素を削除することに成功した。要素の削除のみで追加ができないなど、本来の目標を達成しているとは言い難い部分が多いが、今後、ソフトウェアの改良を重ねることで、ユーザーの負担を大幅に減少できるソフトウェアになると考える。

4.1 今後の課題

- 対応要素の追加。
  - 現在文法に不正があったとしても削除しか行えない。より柔軟で確実な校正をするためには代替要素への書き換えや正しい要素の追加がなくてはならない。
- 属性に関する、判断と処理。
  - 非推奨要素の削除に対応しているのは要素名だけだが、属性とセットになることで非推奨要素となる要素も多数存在するこれらを 1 つのセットとして記録し、各要素単体と分別し削除する。
- 複数のファイル選択インタフェイス。
  - 現在ソフトウェアで指定できる HTML ファイルは 1 つだけである。しかし、ウェブサイトは複数のウェブページで成り立っている。よって、複数の HTML ファイルを指定しウェブサイトを一括校正できるようにする。

謝辞

本研究を行うにあたり、御指導及び御協力頂いた大垣 斉講師、藤井 信夫教授、情報安全工学実験室のメンバー及び卒業生の方々に深く感謝の意を表します。

参考文献

[1] W3C http://www.w3.org/
[2] W3C の仕様書等の文書の日本語訳集 http://www.w3.org/Consortium/Translation/Japanese
[3] Another HTML-lint http://htmllint.itc.keio.ac.jp/htmllint/
[4] 日本 Python ユーザ会（PyJUG） http://www.python.jp/Zope

URI は 2009 年 12 月 1 日現在

付録 A　ソースコード

A.1　hontai.py

```
# -*- coding: utf-8 -*-
#!/usr/bin/python
#GUI 部分



from Tkinter import *
from tkFileDialog import *
import Tkinter
import tkFileDialog
import destruction
import sys




root = Tk()
root.option_add("*font", ("FixedSys", 14))
root.title("HTML 校正ソフトウェア")

iDir='/home/'
name = "out"



def load_file():
    global filenames
    filenames = ""
    args = { "initialdir" : "/home/uemori/python",
             "filetypes" : [("HTML ファイル", "*.html")],
             "title" : "テスト"
             }
    filenames = tkFileDialog.askopenfilenames(**args)



def select_dir():

    global dirname
    dirname = ""
    dirname=tkFileDialog.askdirectory(initialdir=iDir)
```

```
def callback():
    global name
    name = ""
    name = savename.get()


botton01 = Button(root, text = "校正するファイルを選択",
                  command = load_file)

botton02 = Button(root, text = "保存先選択",
                  command = select_dir)

# エントリー
savename = Entry(root, textvariable = buffer)

botton03 = Button(root, text = "出力ファイル名確定",
                  command = callback)

botton04 = Button(root, text = "実行",
                 command = lambda : destruction.
                  item(filenames[0], dirname, name))



la01 = Label(root, text='校正するファイルを選択してください。')
la02 = Label(root, text='保存するディレクトリを選択してください。')
la03 = Label(root, text='出力するファイル名を選択してください。')
la04 = Label(root, text='この設定で実行しますか？')



argvs = sys.argv  # コマンドライン引数を格納したリストの取得
argc = len(argvs) # 引数の個数



if len(argvs) > 1:
    destruction.item(argvs[1],argvs[2],argvs[3])
else:

    la01.pack()
    botton01.pack()
    la02.pack()
```

```
    botton02.pack()
    la03.pack()
    savename.pack()
    botton03.pack()
    la04.pack()
    botton04.pack()
    root.mainloop()
```

A.2 destruction.py

```
#coding:utf-8

def item(fo, fs, name):
       f1 = open(fo, 'r')
       f2 = open(fs + '/' + name + ".html", "w")


       data=[]
       for line in f1:
           stripline = line.strip()
           sent = stripline.replace("<", "\n<")
           sent2 = sent.replace(">", ">\n")
           itemlist=sent2[:-1].split("\n")


           for item in itemlist:
               data.append([item,0])


#---------非推奨要素削除-------------

       moji = ["<applet","<basefont","<center",
               "<dir","<font","<isindex","<listing",
               "<menu","<plaintext","<s",
               "<strike","<u","<xmp",
               "</applet>","</basefont>","</center>",
               "</dir>","</font>","</isindex>",
               "</listing>","</menu>","</plaintext>",
               "</s>","</strike>","</u>",
               "</xmp>"]

       for i in range(0, len(data), 1):
           for j in range(0, len(moji), 1):
```

```
                if moji[j] in data[i][0]:
                    data[i][0] = ""

#---------非推奨要素削除-------------


#---------ブロック要素削除-------------

        kinsi_block = ["<address", "<blockquote","<dir","<div",
                       "<fieldset","<form",
                       "<h1", "<h2", "<h3", "<h4", "<h5", "<h6",
                       "<hr","<isindex","<menu","<noframes","<noscript",
                       "<p","<pre","<table","</address", "</blockquote",
                       "</dir","</div","</fieldset","</form",
                       "</h1", "</h2", "</h3", "</h4", "</h5", "</h6",
                       "</hr","</isindex","</menu","</noframes","</noscript",
                       "</p","</pre","</table"]




#<address>の処理


        flg = 0
        ele = "address"
        for i in range(0, len(data), 1):
               if "<" + ele  in data[i][0] and flg == 0:
                      data[i][1] = "x"
               if "<" + ele in data[i][0]:
                      flg += +1
               if "</" + ele in data[i][0]:
                      flg += -1
               if "</" + ele in data[i][0] and flg == 0:
                      data[i][1] = "y"
               if "</body>" in data[i][0]:
                      break

        count = 0
        for i in range(0, len(data), 1):
               if "x" == data[i][1]:
```

```
                        count += 1

#       kinsi = []

        n = 0
        while n < count :
                for i in range(0, len(data), 1):
                        if "x" == data[i][1]:
                                data[i][1] = 0
                                j = i + 1
                                while "y" != data[j][1]:
                                        for word in kinsi_block:
                                                if word in data[j][0]:
                                                        data[j][0] = ""
                                        j += 1

                        elif "y" == data[i][1]:
                                data[i][1] = 0
                                break

                n += 1
        for i in range(0, len(data), 1):
                data[i][1] = 0
```

#<address>の処理

#ここから<p>の処理

```
        flg = 0
        ele = "p"

        for i in range(0, len(data), 1):
                if "<" + ele in data[i][0] and flg == 0:
                        data[i][1] = "x"
                if "<" + ele in data[i][0]:
                        flg += +1
```

```
                if "</" + ele in data[i][0]:
                        flg += -1
                if "</" + ele in data[i][0] and flg == 0:
                        data[i][1] = "y"
                if "</body>" in data[i][0]:
                        break


        count = 0
        for i in range(0, len(data), 1):
                if "x" == data[i][1]:
                        count += 1

        n = 0
        while n < count :
                for i in range(0, len(data), 1):

                        if "x" == data[i][1]:
                                data[i][1] = 0
                                j = i + 1

                                while "y" != data[j][1]:
                                        for word in kinsi_block:
                                                if word in data[j][0]:
                                                        data[j][0] = ""
                                        j += 1

                        elif "y" == data[i][1]:
                                data[i][1] = 0
                                break
                n += 1
        for i in range(0, len(data), 1):
                data[i][1] = 0
```

#<p>の処理

 # の処理

```
        flg = 0
```

```
ele = "ol"

for i in range(0, len(data), 1):
	if "<" + ele in data[i][0] and flg == 0:
		data[i][1] = "x"

	if "<" + ele in data[i][0]:
		flg += +1

	if "</" + ele in data[i][0]:
		flg += -1

	if "</" + ele in data[i][0] and flg == 0:

		data[i][1] = "y"

	if "</body>" in data[i][0]:
		break


kinsi = ["<li","</li"]


n = 0
while n < count :
	for i in range(0, len(data), 1):

		if "x" == data[i][1]:
			data[i][1] = 0
			j = i + 1

			while "y" != data[j][1]:
				for word in kinsi:
					if word  in data[j][0]:
						data[j][1] = "z"
				if data[j][1] == 0 and "<" in data[j][0]:
					data[j][0] = ""
				else:
					data[j][1] = 0
				j += 1

		elif "y" == data[i][1]:
			data[i][1] = 0
```

```
                        break
        n += 1

for i in range(0, len(data), 1):
        data[i][1] = 0
```

#ここまで<ol>の処理

 # <dl>の処理

```
flg = 0
ele = "dl"
for i in range(0, len(data), 1):
        if "<" + ele in data[i][0] and flg == 0:
                data[i][1] = "x"
        if "<" + ele in data[i][0]:
                flg += +1
        if "</" + ele in data[i][0]:
                flg += -1
        if "</" + ele in data[i][0] and flg == 0:
                data[i][1] = "y"
        if "</body>" in data[i][0]:
                break


kinsi = ["<dt","</dt","<dd","</dd"]

n = 0
while n < count :
        for i in range(0, len(data), 1):
                if "x" == data[i][1]:
                        data[i][1] = 0
                        j = i + 1

                        while "y" != data[j][1]:
                                for word in kinsi:
                                        if word  in data[j][0]:
```

```
                                        data[j][1] = "z"

                                if data[j][1] == 0 and "<" in data[j][0]:
                                        data[j][0] = ""
                                else:
                                        data[j][1] = 0
                                j += 1

                        elif "y" == data[i][1]:
                                data[i][1] = 0
                                break
                n += 1
        for i in range(0, len(data), 1):
                data[i][1] = 0
```

#ここまで<dl>**の処理**

 # <pre>の処理

```
        flg = 0
        ele = "pre"
        for i in range(0, len(data), 1):
                if "<" + ele in data[i][0] and flg == 0:
                        data[i][1] = "x"
                if "<" + ele in data[i][0]:
                        flg += +1
                if "</" + ele in data[i][0]:
                        flg += -1
                if "</" + ele in data[i][0] and flg == 0:
                        data[i][1] = "y"
                if "</body>" in data[i][0]:
                        break


        kinsi = ["<img","<object","<applet","<big","<small","<sub",
                  "<sup","<font","<basefont","</img","</object",
                  "</applet","</big","</small","</sub",
```

```
          "</sup","</font","</basefont"]

    n = 0
    while n < count :
        for i in range(0, len(data), 1):
            if "x" == data[i][1]:
                data[i][1] = 0
                j = i + 1

                while "y" != data[j][1]:
                    for word in kinsi:
                        if word  in data[j][0]:
                            data[j][1] = "z"

                    if data[j][1] == 0 and "<" in data[j][0]:
                        data[j][0] = ""
                    else:
                        data[j][1] = 0
                    j += 1

            elif "y" == data[i][1]:
                data[i][1] = 0
                break
        n += 1
    for i in range(0, len(data), 1):
        data[i][1] = 0
```

#ここまで<pre>の処理

 # の処理

```
    flg = 0
    ele = "ul"
    for i in range(0, len(data), 1):
        if "<" + ele in data[i][0] and flg == 0:
            data[i][1] = "x"
        if "<" + ele in data[i][0]:
            flg += +1
        if "</" + ele in data[i][0]:
```

```
                flg += -1
        if "</" + ele in data[i][0] and flg == 0:
                data[i][1] = "y"
        if "</body>" in data[i][0]:
                break


kinsi = ["<li","</li"]

n = 0
while n < count :
        for i in range(0, len(data), 1):
                if "x" == data[i][1]:
                        data[i][1] = 0
                        j = i + 1

                        while "y" != data[j][1]:
                                for word in kinsi:
                                        if word  in data[j][0]:
                                                data[j][1] = "z"

                                if data[j][1] == 0 and "<" in data[j][0]:
                                        data[j][0] = ""
                                else:
                                        data[j][1] = 0
                                j += 1

                elif "y" == data[i][1]:
                        data[i][1] = 0
                        break
        n += 1
for i in range(0, len(data), 1):
        data[i][1] = 0
```

#<ul>の処理

#---------ブロック要素削除----------

```
#--------- テーブル要素削除-------------



        table_ele = ["<table","</table","<tr","</tr","<td","</td"]

        place = []
        count_t = 0
        flg = 0
        t_flg = 0
        j = 0
        ele = "table"
        del_word = 0


        for i in range(0, len(data), 1):
                if "<" + ele in data[i][0] and flg == 0:
                        data[i][1] = "x"
                if "<" + ele in data[i][0]:
                        flg += +1
                if "</" + ele in data[i][0]:
                        flg += -1
                if "</" + ele in data[i][0] and flg == 0:
                        data[i][1] = "y"
                if "</body>" in data[i][0]:
                        break


        count = 0
        for i in range(0, len(data), 1):
                if "x" == data[i][1]:
                        count += 1


        place_c = 0
        for i in range(0, len(data), 1):
                for e in table_ele:
                        if e in data[i][0]:
                                place.append([i,place_c])
                if "</table" in data[i][0]:
                        place_c += 1
```

```
n = 0
while n < count :
	for i in range(0, len(data), 1):
		if "x" == data[i][1]:
			data[i][1] = 0
			j = i + 1

			while "y" != data[j][1]:
				if "<" in data[j][0]:

					if "/tr" in data[j][0]:
						t_flg += -10

					elif "/td" in data[j][0]:
						t_flg += -1


					elif "tr" in data[j][0]:
						t_flg += 10

					elif "td" in data[j][0]:
						t_flg += 1


				j += 1

		elif "y" == data[i][1] and t_flg != 0:
			for b in range(0, len(place), 1):
				if n == place[b][1]:
					del_word = place[b][0]
					data[del_word][0] = ""

		elif "y" == data[i][1]:
			data[i][1] = 0
			break
	n += 1
for i in range(0, len(data), 1):
	data[i][1] = 0
```

<table>の処理

#--------- テーブル要素削除-------------

```
        j = range(0, len(data), 1)
        for i in j:
                if data[i][0] == "" or data[i][0] == "\n":
                        del data[i]
                        del j[-1]
```

#表示

```
#        for i in range(0, len(data), 1):
#                print data[i]
```

```
        html = ""
        for i in range(0, len(data), 1):
            html += data[i][0]
            html += "\n"

        f2.write(html)

        f1.close()
        f2.close()
```