古い HTML の再利用について

——CSS 化ツールの制作——

0348085　成田　有希

目　次

——古い HTML の再利用について——

1　序論

(1) はじめに

我々、福田洋一ゼミでは「人の役に立つものを作る」がテーマである。

私の制作においての主題は「古い HTML の再利用法」である。これは HTML ファイルの中にデザインタグを書いてしまっている人のために、CSS のデザイン要素を含んでいる HTML ファイルを HTML [(1)]のファイルと CSS [(2)]のファイルの二つに分離して書き出すというものである。

(2) 現状

インターネットに表示されているページは Web ページを記述するためのマークアップ言語[(3)]が使用されている。その中でも HTML 言語は、論理構造や見栄えなどを記述するために使用される言語である。また、文書の中に画像や音声、動画、他の文書へのハイパーリンクなどを埋め込むこともできる。

HTML4.0 以前は文章の論理構造を記述する言語とされてきた。しかし Web ブラウザメーカーにより、見栄えを記述するタグ[(4)]が大量に取り込まれた。それは文書の論理構造を記述する本来の目的に反するため、HTML 4.0 以降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、見栄えの記述は CSS を使って行なうように改められた。CSS を使うことによってフォントや文字飾りなど見栄えに関する情報が切り離されるのである。

(3) 制作に到るまで

三回生の前期に我々、福田洋一ゼミでは、「人に役に立つものを作る」の一環として、人文情報学科のホームページを制作した。今までは HTML の中にデザインタグも一緒に書いてきたが、制作に入る前に HTML と CSS について学び、HTML と CSS に分離して制作することを知った。

HTML4.0 以前に制作したものは、HTML のなかに文字の大きさや背景などデザインに関するタグが同じ HTML の中に書かれている。しかし、HTML4.0 以降は HTML と CSS を分離して制作するようになっている。HTML4.0 以前の HTML の中に組み込まれたデザインに関するタグを CSS に書き換えたほうが、デザインのみを変更したい場合に CSS のみを書き換えれば良いことになる。またページ全体に統一感をだすことができ、合理的に制作していくことが出来るのである。

しかし、以前に制作した HTML を HTML と CSS に分けようとすると書き変えに時間がかかってしまう。そこで HTML4.0 以前に制作した HTML を書き変える時間を最小限に抑えるために、HTML と CSS に自動的に変換できないかと考えた。もし、手動で変換することになると、HTML タグが CSS ではどのタグになるのか、対応するものを探していかなければならない。

自動的に変換することによって、HTML と CSS を分けて書き変えようとするときに、時間を省略することができ、少しでも書き換えがやりやすくなるのではないか。また、そのようなサイトがオンライン上にあれば、誰もが使用することができ、もっと便利だろうと考えた。サイトがあれば検証を行い、より使いやすいものにするはずだったが、見つからなかったので自分で制作することにした。

2　制作過程

(1) プログラムの方針

制作に入る前に、HTML のタグをどのように CSS のタグに変換していくのかを検討した。

例えば、HTML の中に<body bgcolor=”red”>と記述しているタグがある。HTML では ‘bgcolor’ という属性名[(5)]を、CSS では ‘background-color’ という属性名に変換しなければならない。‘bgcolor’ という属性名を ‘background-color’ に置換し、‘red’ という属性値をを取り出して、CSS では body { backgroundcolor: red; } と書き換えることが可能ではないかと推測した。

<body bgcolor=”red”>では、タグとは ‘body’、属性名とは ‘bgcolor’、その値である ‘red’ が属性値になり、デザインを記述するタグとなる。変更するときに、タグの ‘body’ と属性値の ‘red’ はそのまま CSS に移行し、HTML の中に ‘bgcolor’ と書かれている属性名を検索し、CSS では、属性名を ‘backgroud-color’ に置換することができると考えた。

以上のことから HTML と CSS を分けて書き変えることのできる支援ツールを制作することができるのではないかと思い、制作することを決めた。

(2) 制作するにあたって

制作に必要な技術として、Python [(6)]を使ってプログラムを制作する。しかし、プログラムの中はほとんどタグや属性名を検索し、置換するというものなので、正規表現[(7)]を使用する。

本は HTML と CSS のタグの違いを学ぶためには『最新実用 HTML タグ辞典』 [1] を、Python を学ぶためには『Python で学ぶプログラム

作法』 [2] を正規表現には『Web プログラマのための正規表現実践のツボ』 [3] を使用した。

また、制作にとりかかる前に、HTML と CSS の違いについてわかっておかなければならない。まずは Excel で表（資料 HTML と CSS の違い.xls 参照）を作成し、HTML と CSS の違いを知ったうえで、プログラムを制作していくことにした。表の項目には、HTML 開始タグと HTML 終了タグ、CSS 開始タグと CSS 終了タグ、タグの意味の５項目を設けた。

HTML に<body text = "...">という本文の文字の色を指定する開始タグがある。この HTML 開始タグを CSS 開始タグでは、body { color: ...} と変換しなければならない。また HTML 開始タグに対応する終了タグは、</body>である。この HTML の</body>を CSS 終了タグでは、</style>と変換する。このようなタグの場合、HTML と CSS ではタグ名は同じで属性名が変わるものということになる。

HTML に<hr color = "...">という水平線の色を指定する開始タグがある。この HTML 開始タグを CSS 開始タグでは、hr { color: ... ;} と変換する。このタグには、HTML 開始タグに対応する終了タグも CSS 終了タグもないので、HTML と CSS の開始タグのみを変換する。このようなの場合、HTML と CSS ではタグ名と属性名が同じものということになる。

HTML にという文字を太くする HTML 開始タグがある。この HTML 開始タグを CSS 開始タグでは、span { font-weight: …} と変換しなければならない。また HTML 開始タグに対応する終了タグは、である。この HTML 終了タグを CSS 終了タグでは、</style>と変換する。このようなタグの場合、タグ名・属性名ともに全て変換す

る。

そして、タグの中には、<ul type=”...”>のように属性名が一つのものと<body bgcolor=”...” text=”...”>や<table width=”...” height=”...”>のように属性名が二つ以上のものがある。

このように、タグを確認しながら HTML と CSS の違いについて調べていった。

(3) 制作環境

制作は Windows のサクラエディタを使用した。制作したものについては、コマンドプロンプト(8)で実行、テストし、問題点をサクラエディタで変更していくという作業である。

しかし、実行する前に、まず HTML の文法を確認してもらいたい。

そのためには、Another HTML-lint gateway（“http://htmllint.itc.keio.ac.jp/htmllint/htmllintl.html”）で、URL・DATA・FILE のどれかでチェックをしてから、90 点以上の HTML ファイルのみに変換ツールを使用していただきたい。そうすることによって、HTML の文法も確認でき、このプログラムのエラーも少なくなるからである。

このプログラムを Windows で実行する場合、あらかじめ Python（“http://www.python.jp/Zope/download/pythoncore”）をインストールしておく必要がある。

そしてコマンドプロンプトを開き、コマンドライン(9)でプログラム本体のファイル名 result.py のあとに半角スペースを空け、実行するファイル名を一つ指定し、Python を実行させる。(z:\>pyhon result.py test.html)

Enter キーを押して、エラーなく z:\>がかえってくれば変換できて

いる。しかし、エラーがあり、z:\>がかえってこなければ、もう一度ファイル名の綴りなどを確認してやり直す。

エラーなく z:\>がかえってくれば、実行した HTML ファイルがある同じフォルダに style.html を style.css という二つのファイルが自動的に表示されている。style.html は変換前の HTML ファイルを CSS と別にするために変換したものであり、style.css は変換前の HTML ファイルに書かれていたデザイン要素を style.css に書き換えたものである。

3　CSS 化ツールの制作

■　プログラムの構成

- 最初の宣言文（資料 result.py 1 行目～12 行目・325 行目参照）
- 属性名や属性値を変更するテーブル（資料 result.py 15 行目～46 行目参照）
- 正規表現をコンパイルして変数に代入している部分（資料 result.py 50 行目～61 行目参照）
- 変換する HTML ファイルの検索と読み込み（資料 result.py 50 行目、217・218 行目参照）
- 関数定義（資料 result.py 65 行目～211 行目参照）
  - ○ 属性名が一つの場合
  - ○ ファイルの中のタグを全て小文字に変換する場合
  - ○ <body>に二つ以上の属性名がある場合
  - ○ <body>の変換
  - ○ 属性名が二つ以上ある場合
  - ○ テーブルに関するタグに属性名がある場合
  - ○ フォントタグに属性名がある場合

○ 属性名が二つ以上あるタグの変換

・HTML と CSS でタグ名が変わるものの変換（資料 result.py 275 行目～299 行目参照）
・前処理（資料 result.py 221 行目～253 行目参照）
・文字コードを変換する部分（資料 result.py 255 行目～271 行目参照）
・正規表現で置換し、書き出す部分（資料 result.py 273 行目・274 行目・302 行目～310 行目参照）
・style.html と style.css の二つのファイルを書き出す（資料 result.py 314 行目～324 行目参照）

■ 最初の宣言文

プログラムのはじめには Python を使用するために宣言しておく必要がある。また、import sys, re のモジュールは関数や正規表現の使用を可能にするためのものである。

また HTML4.0 以降のものを定義するために、<!DOCTYPE HTML PUBLIC "-//.....">を変数 “DOCTYPE” に代入しておく。

そして、結果を代入していく変数 “html”、“css”、“css2” を用意しておき、“classno” を “0” に設定しておく（資料 result.py 1 行目～12 行目参照）。

■ HTML と CSS でタグ名が変わるものを変換

HTML4.0 以前は<b>と書かれてきたものが、HTML4.0 以降では、CSS では span.bold { font-weight: bold } と書くようになった。このように、HTML と CSS でタグ名・属性名ともに全く違う書き方をするタグについては、HTML では、CSS では span.bold { font-weight: bold } というよう

に単純に変換するだけにした。

そのようなタグには、、、<u>、<s>、<strike>、<brink>、、 が挙げられる。、、<u>、<s>など、HTML4.0 以前のタグを HTML と CSS でそれぞれのタグ名と属性名に変換する（資料 result.py 275 行目～299 行目参照）。

例えばが、変換する HTML の中にあれば、HTML ではに、CSS では、span.bold { font-weight: bold } に変換する（資料 result.py 276 行目～278 行目参照）。

また、このような HTML4.0 以前に書かれたやのようなタグには、終了タグであるやは必須であるため、まずそれぞれの終了タグを検索し（資料 result.py 55 行目参照）、終了タグがあれば、で置換する（資料 result.py 302 行目参照）。

のみは、 を使用した。同様のタグに があるが はインライン要素（前後に改行が入らない）として、 はブロック要素（前後に改行がはいる）として定義されているため、 については、変更する HTML ファイルに があれば、 に置換するようにした（資料 result.py 297 行目～300 行目参照）。

■　HTML の属性名を CSS の属性名を変更するテーブル

HTML では ‘bgcolor’、CSS では ‘background-color’ のように HTML と CSS では同じデザイン要素の属性名でも書き方が変わってくる。

そこで、Excel で作成した表の HTML と CSS の属性名を見比べながら、HTML の属性名を CSS の属性名に変更するテーブル “attr_table” を作成した（資料 result.py 21 行目～46 行目参照）。

“attr_table” のキーと値はどちらも文字列でなければならないこと、さらにキーの参照時に大文字と小文字を区別しないことを前提とし、例えば、“bgcolor”:“background-color”、“align”:“text-align” のように属性名が異なるものを、“attr_table” にまとめる。

■　正規表現のコンパイル（検索）と置換

変換する HTML ファイルの中には、それぞれにいろいろな属性名や属性値を持ったタグが多く存在している。一つ一つのタグに関して、検索し、置換しているとプログラムが大きなものになってしまうので、いくつかのグループに分けてコンパイルと置換をすることにした。コンパイルする場合、日本語の正規表現を使用するために文字コードは unicode(10)に直す。

■　正規表現でファイル名のコンパイルと置換

変換するファイルが HTML のファイル名（〜.html または〜.htm）かどうか検索するために、コマンドラインの引数(11)で指定したファイル名があるかを re_file_name = re.compile(u’\.html?$’) の正規表現でコンパイルする（資料 result.py 50 行目参照）。この正規表現は、何かの文字列の行末に ‘.html’ があれば、文字コードを unicode に直し、’“re_file_name” に代入する。

input_file_name = sys.argv[1] でコマンドラインの引数で指定したファイル名を変数 “input_file_name” に代入する（資料 result.py 218 行目参照）。

置換の正規表現は、output_file_name = re_file_name.sub(”.x.html”, input_file_name) の部分で行う。“input_file_name” に代入したファイル名を、〜.html のファイル名に置換し、“output_file_name” に代入する（資料 result.py 218 行目参照）。

■　正規表現でタグの中を小文字に変換する

変換する HTML ファイルには、タグの中に大文字と小文字が混在しているものが多く存在している。そのようなファイルを変換するには、小文字と大文字を別々に正規表現で検索し、置換しなければならなかったので、変換する HTML ファイルの中を全て小文字で書き出すことにした。

そのためには、re_tag_lower = re.compile(r'<.+?>') の正規表現で、'<' と '>' 中に改行を除く任意の文字列（.）を 1 回以上繰り返しているもの（+）や 1 回または 0 回繰り返しているもの（?）をコンパイルする。つまりタグがあるかどうかを検索しているのである（資料 result.py 62 行目参照）。

タグがあれば、html = re_tag_lower.sub(tag_lower, html) の正規表現で、関数 "tag_lower" を呼び出し（資料 result.py 65 行目・66 行目参照）、"html" に入っている HTML ファイルのデータを全て小文字に置換し、"html" に代入する（資料 result.py 271 行目参照）。

■　属性名が一つのタグの場合

タグの中の属性名は<ul style=...>のように属性名が一つしかないタグと、<body bgcolor=... background=url(...)>のように属性名が二つ以上のものがある。

ここでは、re_1attr = re.compile(u'<(h1|h2|h3|h4|h5|h6|tt|ul|li)?(([A-Za-z]+) ?= ?"?(\w+)"?)?>') の正規表現で、'(' と ')' の中のタグ名に半角スペースを 1 回または 0 回繰り返しているもので A から Z または a から z の文字列（[A-Za-z]）を 1 回以上繰り返しているものかつ、英数字（\w）とマッチしているものをコンパイルし、"re_1attr" に

代入する。つまり、<h1>～<h6>や<ul>のように属性名が一つしかないタグのみを検索しているのである（資料 result.py 55 行目参照)。

属性名が一つのタグがあれば、html = re_1attr.sub(one_attr, html) の正規表現で、関数 “one_attr” を呼び出し（資料 result.py 70 行目～84 行目参照)、“html” に入っている HTML ファイルのデータを置換し、“html” に代入する（資料 result.py 307 行目参照)。

ここで、使用する “one_attr” では global 文を使用する。これは、関数の外にある変数に値を代入できる働きを持ち、このような変数をグローバル変数と言う。今後、全ての関数に global 文を使用していく。

“one_attr” では置換する場合に、そのタグを取り出し、変数 “tag” に代入しておく。属性名が一つしかないタグがある場合、HTML の属性名を CSS の属性名を変更するテーブル “attr_table”（資料 result.py 21 行目～46 行目）を使用し、変換する属性名があれば、属性名を置換し、変数 “attr” に代入する。もし、“attr_table” に属性名がない場合にも、その属性名を “attr” に代入しておく。

さらに、属性値を変数 “value” に代入し、変数 “classno” を一つ追加する。変数 “newclass” に “tag”（タグ名）と “value”（属性値)、“classno”(ナンバー）を代入し、それぞれ “css” に書き出す。属性名がなければ、“html” にタグ名のみを書き出す。

■　属性名が二つ以上のタグ場合

次に<body>や<table>のように属性名が二つ以上の場合、属性名が一つのものと同じように、属性名が二つ以上あるタグを re_many_attr = re.compile(u’<([0-9a-z]+) (.+?)>’) の正規表現でコンパイルする。<と>の中に 0 から 9 または a から z の文字列（[0-9a-z]）を繰り返しているもので、改行を除く任意の文字列を 1 回以上繰り返しているものや 1

回または 0 回繰り返しているものにマッチさせ、“re_many_attr” に代入する（資料 result.py 59 行目参照)。

タグに二つ以上の属性名があるならば、html = re_many_attr.sub(replace_many_attr, html) の正規表現で、関数 “replace_many_attr”（資料 result.py 185 行目〜211 行目参照）を呼び出し、“html” に入っている HTML ファイルのデータを置換し、“html” に代入する（資料 result.py 310 行目参照)。

ここで使用する “replace_many_attr” では、属性名が一つの場合と似たような変換をするのだが、いくつか違う点がある。

一つ目に属性名は新しい変数 “properties” に代入することである。“properties” に入っている属性名に’class’ が入っていれば、タグと属性名を書き出す。

二つ目にタグや属性名によって変換方法を変えるのである。“tag” に入っているタグ名が’font’ や’table’ に関するタグならば、また別の関数で処理し、それ以外は、関数 “each_property”（資料 result.py 125 行目〜135 行目参照）を使用し、置換するようにした。

もし、“tag” に ‘font’ があれば、置換するときに関数 “font_each_property”（資料 result.py 164 行目〜181 行目参照）を使用する。属性名は “attr” に、属性値は “value” に代入しておく。属性値が”（ダブルクォーテーション）で始まるものならば、” から” までの属性値を取り出して、“value” に代入しておく。属性名に’style’ があれば、“html” と “css” に属性名と属性値をそれぞれに書き出し、“css” に属性名と属性値を書き出す。。

もし、“tag” に ‘table’ に関するタグがあれば、置換するときに、関数 “table_each_property”（資料 result.py 139 行目〜160 行目参照）を使

用する。属性名は “attr” に、属性値は “value” に代入しておく。属性値が” で始まるものならば、” から” までの属性値を取り出して、“value” に代入する。“attr_table” に変換する属性名があれば、属性名を置換し、“attr” に代入する。“css” に属性名と属性値を書き出す。

■ body タグの変換

<body>は、多くの属性を持っているので、他のタグとは別に変換することにした。

まず、re_body = re.compile(u’<body (.+?)>’) の正規表現で、<body>が変換する HTML ファイルにあるかを検索する。<body>に改行を除く任意の文字列を 1 回以上繰り返しているものや 1 回または 0 回繰り返しているものを検索し “re_body” に代入する（資料 result.py 58 行目参照)。

<body>があれば、html = re_body.sub(replace_body, html) で、“html” に入っている HTML ファイルのデータを関数 “replace_body”（資料 result.py 112 行目〜121 行目参照）で読み込み、置換し、“html” に代入する（資料 result.py 274 行目参照)。

また、<body background=url(...)>のように属性名があれば、“properties” に属性名を代入し、関数 “body_each_property”（資料 result.py 88 行目〜108 行目参照）を使って置換し、“css” に書き出す。属性名がなければ、<body>のみを “html” に書き出すようにした。

この場合に使用する “body_each_property” は、属性名があれば、その属性名と属性値をそれぞれ “attr” と “value” に代入しておく。

属性値が” で始まるものならば、” から” までの属性値を取り出して、“value” に代入する。“attr” が’link’、’alink’、’vlink’ があれば、属性名を変更し、それぞれ “css” に書き出す。

次に、re_endbody_tag = re.compile(u'</body>') の正規表現で</body>を検索し、“re_endbody_tag” に代入する（資料 result.py 56 行目参照)。</body>は、CSS を読み込むために</style>に置換するからである。</body>があれば、html = re_endbody_tag.sub('</style>', html) の正規表現で、“html” に</body>があれば、</style>に置換し、“html” に代入する（資料 result.py 303 行目参照)。

■ style.html と style.css の書き出し

プログラムの最後に、これまで変数に代入してきたデータを style.html と style.css を書き出さなければならない。

まず style.html については、“head” や “html” に代入してきたデータの文字コードを unicode から変換前の HTML ファイルの文字コードに戻しておく（資料 result.py 320 行目〜321 行目参照)。

style.html を書き込みモードで open し、“output_file_name” に代入する。これまで、“head” や “html” に代入してきたデータを全て “output_file_name” に書き出し、ファイルを表示する。

次に style.css は style.html と同様に処理をする。“css” に代入してきたデータの文字コードを unicode から変換前のファイルの文字コードに戻しておく。style.css を書き込みモードで open し、“css_file” に代入する。これまで、“css” に代入してきたデータを “css_file” に書き出し、ファイルを表示する（資料 result.py 313 行目〜323 行目参照)。

4 結論

2006 年 11 月 1 日の時点で制作途中のプログラムを人文情報学科の学生 5 人に使用してもらい、良かった点と問題点を改善するためにテストして貰い、アンケートを行なった。

良かった点として、全員がきちんとファイルが HTML と CSS に分かれていたこと、操作性の良さの評価を頂いた。

しかし、問題点としては、きちんと変換できておらず、レイアウトが違うという評価を多く貰った。また、変換すると全く表示されなくなったという意見もあった。レイアウトが違うという問題点は、フォントサイズが違う、改行やスペースが表示されない、CSS に変換しないものまで変換されている、背景画像が挿入されていない、テーブルの大きさが違う、テーブルの枠線が表示されていないという点があった。

また、変換すると全く表示されなくなったものに関しては、文字コードの問題であると判断し、文字コードについての処理方法をプログラムに書き直した。

アンケートで使用したファイルは一つずつテストし、問題点を以下のように改善した。

(1) 問題点と考察

(i) フォントサイズの調整

プログラムの中にフォントサイズの処理方法を書いていなかったので、HTML ファイルを変換すると、フォントサイズが反映されていなかった。

HTML でフォントサイズを指定する場合、 1〜7 の数値を指定する。既定値は通常 “3” で、size=”+1” や size=”-1” のように、現在の大きさに対する相対値を指定することもできる。CSS では、絶対指定として xx-small、x-small、small、medium（既定値)、large、x-large、xx-large、絶対単位指定として 10in、10cm、10mm、10pt、10pc、相対単位指定として 10px、10ex、10em などを割合指定として 120% など

を指定する。

HTML と CSS のフォントサイズを同じ大きさにするために、プログラム本体にフォントサイズを変更するテーブル “font_size_table”（資料 result.py 15 行目〜18 行目参照）を作り、対応させて変換することにした。

HTML では通常 “3” のフォントサイズを CSS では “medium” に設定し、“1” は “x-small”、“2” は “small”、“4” は “large”、“5” は “x-large”、“6” は “xx-large” と設定した。しかし、“7” のサイズ設定が絶対指定になるとなくなってしまうので、HTML と CSS のフォントサイズを見比べながら、“150%” に合わせた。

“font_size_table” は、例えば HTML に<font size =”4”>のように、属性名に ‘size’ があれば、“font_each_property” を呼び出し、属性名の ‘size’ を’font-size’ に変換する。そして、“font_size_table” に属性値のキー（ここでは “4”）があれば、値を “large” に変換する。しかし、属性値のキーがなければ、規定値の “medium” に変換することにした。

（ii）改行・スペースの調節

変換前の HTML ファイルに改行が挿入されていなかったり、中途半端の位置で改行が入っていたりすると、うまく変換できない。またエラーなく変換されたとしても、表示してみると変換できていないことがあり、違うレイアウトになっていた。

原因は変換する前のファイルの改行がタグと属性の間で入っていたり、属性と属性値の間に入っていたりしたためである。

そのため、まず re_block_tag = re.compile(r’</(p|blockquote|ol|ul-|dl|table|li|caption|dd|dt|head|td|tr|div|tbody)>’) の正規表現で、‘(’ と ‘)’ の中のタグ名とマッチしているものを検索し、

“re_block_tag” に代入する（資料 result.py 50 行目参照）。つまり、</p>や</tr>、</td>のようなブロックタグの終了タグを検索するのである。終了タグがあれば、html = re_block_tag.sub('\g<0>\n', html) で変換したときに終了タグの後ろで改行を挿入する（\g<0>\n）ように置換した。“\g<0>\n” は ‘(’ と ‘)’ で囲まれたグループをそのまま書き出すようにする命令である（資料 result.py 304 行目・305 行目参照）。

また、変換後の style.html で、<hr class=”hr_1”>と変換されなければいけない部分が、タグ ‘hr’ と属性名 ‘class’ の間に半角スペースが二つ入っていて、<hr　class=”hr_1”>と変換されていたため、CSS には反映されなかった。

そこで、re_spaces = re.compile(r’ {2,}’) で、半角スペースが二つ以上繰り返している（{2,}）文字列を検索し、“re_spaces” に代入する（資料 result.py 54 行目参照）。終了タグがあれば、html = re_spaces.sub(’ ’, html) の正規表現で半角スペース一つだけに置換し、“html” に代入する（資料 result.py 306 行目参照）。そうすると、<hr class=”hr_1”> のようにうまく変換することができた。

（iii）リンク、入力フォームなど CSS に変換しないタグ

リンク～や入力フォーム<form action=”...”>～</form>などには一般属性はあるが、デザインタグ<b>のように見栄えに関する記述はない。

プログラム上でこのまま変換すると、HTML には<a =”a_1”>、CSS には a { href=”...” } と表示されてしまう。HTML・CSS ともに、このような記述のしかたはない。

そこで、属性名を二つ以上持つタグの場合 “replace_many_attr” に ‘a’、‘form’、‘img’、‘input’、‘script’ など変換しないタグがあれば、変

換せずにそのままの状態でタグと属性名を書き出すことにした。

（iv）背景画像の挿入

<img src="...">は先ほど述べたように CSS には変換しないタグであるが、同じ画像でも<body>に背景画像を挿入したいとき、または<table>に背景画像を挿入したいときに、HTML では<body background="test.jpg">、または<table background="test.jpg">と記述する必要がある。

しかし、style.css に変換した場合、body { background-image: "url(test.jpg)"; } と変換されてしまうために、背景画像が表示できなかった。CSS には body { background-image: url(test.jpg); } と変換したいため、属性値である背景画像のファイル名をそのまま変換してしまうと、" も CSS に反映されるという問題が生じた。

そのため、"body_each_property" を呼び出し、変換する HTML ファイルの<body>に<body background="test.jpg">と背景画像を挿入する属性名があるか検索し、あれば属性名を "attr" に代入する。属性名の'background' を'background-image' に変換し、そして属性値のみを "value" に代入して、'url' と一緒に "css" に変換するようにした。

<table>に背景画像を挿入したいときも、同様に "table_each_property" を呼び出し、置換することによって、style.css には table { background-image: url(test.jpg); } と変換された。

（v）文字コードによるエラー

変換前の HTML ファイルは様々な文字コードで書かれている。また、'charset=' の部分が、はじめから宣言されているものと宣言されていないものがある。

宣言されている場合、主な文字コードには Shift_JIS[12]、euc-jp[13]、

iso-2022-jp(14)、UTF-8(15)が挙げられる。

はじめに文字コードは何で書かれているか ‘charset=’ の部分を検索する（資料 result.py 61 行目参照）。その文字コードが<head>の中で宣言されているかいないかを検索し（資料 result.py 255 行目参照）、文字コードが宣言されているならば、その文字コードを変数 “charset” に代入する（資料 result.py 256 行目・257 行目参照）。文字コードが宣言されているのに識別できないのであれば、ユーザーに文字コードは何を使用しているか尋ね、ユーザー自身に文字コードを確認してもらうことにした（資料 result.py 258 行目～268 行目参照）。しかし、文字コードがはじめに宣言されていなければ、強制的に ‘charset=Shift_JIS’ として書き出すようにした。

このように変換することで、変換後と全く表示されなくなったものが表示できるようになった。

また、ここまでの文字コードは全て unicode で変換し代入していたが、ここで “head” と “html” を変換前の文字コードに変換しなおし、最後に書き出す（資料 result.py 270 行目・271 行目参照）。

（vi）table の問題

表を挿入している HTML 文書には<table>タグが組み込まれている。例えば、HTML で<table width=“500” height=“500”> と書いているタグある。CSS では table {width: 500px; height: 500px; } と表示するので、‘width’ や ‘height’ など横幅や高さは単位付きの数値で指定しなければならない。

しかし、変換すると ‘px’ の単位がなくなり、table {width: 500; height: 500; } のように数値だけ CSS に書き出されるので、style.css にデザインが反映されていなかった。

そこで “table_each_property” で、属性値のはじまりに、” があるかを検索する。” があれば、” から” まで属性値の数値のみを取り出して、“value” に代入する。属性名に ‘width’、‘height’ が変換する HTML ファイルにあれば、“value” という変数に、” をはずして、属性値と ‘px’ の単位を付けて、“css” に書き出すことにした（資料 result.py 139 行目～160 行目参照）。

そうすることにより、style.css には単位付きで table {width: 500px; height: 500px; } と書き出され横幅や高さが正確に表示されるようになった。

（vii）table-border の問題

また HTML では、枠線の太さは ‘1px’、枠線のスタイルは実線で、枠線の色は黒く表示するには<table border=1>とすれば表示される。

しかし、CSS で枠線の太さを ‘1px’、枠線のスタイルは実線で、枠線の色は黒で表示するには table {border: 1px solid black; }、または table {border-width: 1px; border-style: solid; border-color: black; } と書かなければ、枠線のスタイルや色は表示されない。

そこで前項と同じように、まず “table_each_property” で、属性値のはじまりに” があるかを検索する（資料 result.py143 行目参照）。” があれば、” から” までの属性値の数値のみを取り出して、“value” に代入する。変換する HTML ファイルに ‘border’ があれば、“value” に、” をはずして、属性値と ‘px’ の単位を付ける。属性名の ‘border’ を “border-width” に変換し、“attr” に代入し、“css” に書き出すことにした（資料 result.py 139 行目～160 行目参照）。

その際、テーブルやセルのそれぞれに枠線が表示されるように、“replace_many_attr” 関数を使用し、“tag” に ‘table’ があれば、css に

table, td, th {border: 1px solid black; } と書き出されるように設定した。

しかし、そこでまた一つ問題点が起こり、<table border=0>と枠線の非表示が設定されている場合、どうするのかという問題点が残った。

（viii）HTML の文法の問題

このツールを実行するときに気をつけてもらいたいのが、HTML の文法の問題である。ユーザーが使用している HTML の文法は正しいものばかりとは限らない。変換前の HTML ファイルの文法が間違っているものが多く、エラーがでていた。

今回のテストで、一番多かった文法のエラーは、<td align=middle>という文法である。本当ならば、この文法は、<td align=center>[16]と書くか、<td valign=middle>[17]と書くのが正しい。

このように、少しでも文法が間違っていれば、変換できなくて、style.css が違ったレイアウトになってしまうということがある。

今回のエラーに関しては、あまりにも<td align=middle>と記述している HTML ファイルが多くあったので、“table_each_property” で、属性名が’align’、属性値が’middle’ であれば、属性名を’vertical-align’ に変えて、“attr” に代入し、“css” に書き出すことにした。属性名を’vertical-align’ に変換することで、属性値を変更せずに書き出すことに成功した（資料 result.py 155 行目・156 行目参照）。

そのため、前述でも述べたが、この変換ツールを使用する前に HTML 文法チェックで最低でも９０点以上を取ってから、このツールを使ってもらったほうがエラーも少なくでき、変換前の HTML ファイルも正確に記述することができるのではないだろうか。

(2) 自己評価

今回は『古い HTML の再利用について－ CSS 化ツールの制作－』という題で、制作を行った。

今までは、私自身も HTML4.0 以前を活用することが多かったので、HTML と CSS に分離するという考え方すら知らなかったが、今回の制作を通して私のような人が少しでも、このツールを使用していただければと思う。

序論でも述べたが、HTML と CSS を分けて制作することで、多くの HTML ページを制作する上では、HTML のみページを増やしていき、あとは CSS を<link rel=”stylesheet” type=”text/css” href=”style.css”>で読み込めば、同じようなデザインのレイアウトで表示することができる。HTML4.0 以前のものを使用してきた人にとっては、今までの制作したものを HTML と CSS に分離できればいいと考えるだろう。このような人に役立ててもらうために制作した今回のツールは、自動的に HTML と CSS に分離できるため、効率的に早く変換できる。

そのため HTML4.0 以前に制作した HTML ファイルを残したまま、CSS に変換することができたので、HTML ファイルを再利用できたと言える。そのような点では制作が実現でき、HTML4.0 以前の古い HTML も再利用できるという見解に辿りついた。

しかし、できるだけユーザーが使用しやすいように制作するのが目標だったが、その点では、まだまだ改善点がある。制作する前までは、Another HTML-lint gateway のように、Web 上で誰もがアクセスでき、またチェックにかけて、エラーがでたものに関してはエラーの内容も表示させるつもりだった。

しかし、プログラムを組み、いくつかの Web ページをテストで変換してみると、次々にエラーが発生し、改善しても、同じページでまた違うエラーが発生するなど、一つのページの問題点を改善するのに時間がかかってしまった。結果、コマンドプロンプトで変換し、ユーザー自身で開いてもらうという半分手動のようなものになってしまった。

また、できるだけ問題点を少なく、同じようなレイアウトになるようにプログラムを組んできたわけだが、一つのファイルをテストするのに時間がかかりすぎて、多くのファイルをテストできなかった。本当ならば、もっと問題点が生まれてくるはずだが、テストしたファイルが少なかったため、できるだけ多くの問題点を発見し、改善することができなかった。

(3) 今後の改善点

今後の改善点としては、　Another HTML-lint gateway のように、Web 上で誰もがアクセスできるようにすること　table-border の問題で、<table border=0>と枠線の非表示が設定されている場合、枠線が表示されないようにすることが挙げられる。

　に関しては、アクセスできても実際に使用できなければ、意味がないので Python のこのプログラムをインターネット上で使えるように cgi[18]に書き換えることが必要である。そしてユーザーが、URL・DATA・FILE のどれかでチェックをするために、HTML のページを作成し、送信ボタンを押してプログラムを起動できるようにしたい。また、変換結果を画面上に書き出して、ユーザーがいち早く見ることができるように改善していきたい。

　に関しては、新しく関数定義をして、条件分岐しなければならない。

もし、属性名に table-border があった場合、‘border’ の属性値が “0” なのか “1” 以上なのかで処理をどうするか考察する。“0” の場合は枠線を非表示にするため、“css” に table, td, th {border: 1px solid black; } と書き出さず、“css” には何も処理しないで変換する。‘border’ の属性値が “1” 以上ならば、table-border の問題でも述べたように、テーブルやセルのそれぞれに枠線が表示されるように、“css” に table, td, th {border: 1px solid black; } と書き出すように設定するとうまく変換することができるのではないだろうか。

(4) 最後に

今回、プログラムを組むことで「人の役に立つものを作る」ためには、簡単に動けば良いわけではなく、ユーザーが使いやすく、わかりやすいものを作らなければならないということを理解した。

最後に、このツールをできるだけ多くの人に使用していただければと思う。今後もさらなるツールの発展に向けて研究していく中で、改善を繰り返し、よりよいものを作っていかなければならない。

注

（1）HTML=Hyper Text Markup Language
Web ページを記述するためのマークアップ言語

（2）CSS=Cascading Style Sheets
Web ページのレイアウトを定義する規格

（3）文書の一部を「タグ」と呼ばれる特別な文字列で囲うことにより、文章の構造や見栄えの記述を文章中に記述していく記述言語

（4）**「<」と「>」で囲まれた標識**

（5）HTML のタグには bgcolor=red や size=5 などの属性（attribute）を指定することができる。color の部分を属性名、red の部分を属性値と呼ぶ。

（6）プログラミング言語のひとつで、コンパイルを必要としないスクリプト言語

（7）文字列のパターンを表現する表記法。正規表現を使えば、文字列を直接指定せず、「特徴」(パターン) を指定することができる。

（8）MS-DOS や UNIX 系 OS などキーボードから文字で命令を入力して操作を行なう。システムが命令入力を受け付けられる状態にあることを示すために表示される記号。ユーザーはこれに続けてコンピュータへの指示を入力し、Enter キーなどで決定する。

（9）画面上に命令の入力を促すプロンプトと呼ばれる文字列が表示され、ユーザーがそれに続けてキーボードからコマンド (命令) を入力し、コンピュータに指示を与える。

（10）すべての文字を 16 ビット (2 バイト) で表現し、1 つの文字コード体系で多国語処理を可能にしようとするもの。世界の主要な言語のほとんどの文字を収録している。

（11）プログラム中で関数やメソッドなどを呼び出すときに渡す値のこと

（12）日本語文字コードの一つ

（13）日本語だけでなく複数バイト言語の各国の文字コードが規定されている。日本語の EUC コードを特に「EUC-JP」「日本語 EUC」と呼ぶこともある。

（14）JIS 規格によって規定されている日本語の文字コードの一つ。Shift_JIS コード、日本語 EUC と並んでインターネット上でよく使われる文字コードである。

（15）1 文字を 1〜6 バイト (現状では最長 4 バイト) の可変長の数値 (バイト列) に変換する

（16）“http://www.tohoho-web.com/html/td.htm” より

（17）“http://www.tohoho-web.com/html/td.htm” より

（18）cgi=Common Gateway Interface

Web サーバが、Web ブラウザからの要求に応じて、プログラムを起動するための仕組み

参考文献

[1] 古籏 一浩『最新実用 HTML タグ辞典』. 技術評論社 , 2000

[2] Alan Gauld『Python で学ぶプログラム作法』. ピアソンエデュケーション, 2001

[3] 高橋 良明『Web プログラマのための正規表現実践のツボ』. 九天社, 2005

資料一覧

Python Japan User's Group 2006　Python Japan User's Group
（http://www.python.jp/Zope/）

Python Japan User's Group 2006　正規表現 HOWTO
（http://www.python.jp/Zope/articles/tips/regex_howto）

Python Japan User's Group 2006　Python チュートリアル
（http://www.python.jp/doc/2.3.5/tut/）

Python Japan User's Group 2006　Python ライブラリリファレンス
（http://www.python.jp/doc/2.3.5/lib/lib.html）

Python Japan User's Group 2006　Python リファレンスマニュアル
（http://www.python.jp/doc/2.3.5/ref/ref.html）

master@jurapun.com 2001　Python ハンディマニュアル
（http://hp.vector.co.jp/authors/VA003670/python/）

杜甫々 2005　とほほの WWW 入門
（http://www.tohoho-web.com/www.htm）

Incept Inc 2006　IT 用語辞典 e-Words
（http://e-words.jp/）

——古い HTML の再利用について——

k16@chiba.email.ne.jp 2006　Another HTML-lint gateway
（http://htmllint.itc.keio.ac.jp/htmllint/htmllintl.html）