Support de cours n°4

DESS TEXTE

Introduction

Ce support de cours donne une description des modules de la librairie HTML, modules qui sont utilisés pour l'extraction d'informations de pages web. Il s'inspire très largement du cours de Fiammetta Namer, responsable du DESS TEXTE de l'Université de Nancy 2.

Modules de HTML

Ici, nous allons regarder de près ce qui concerne l'analyse syntaxique d'une page HTML. Successivement, nous supposerons que (1) l'input est un fichier html (éventuellement issu d'un `getstore`), (2) l'input est une chaîne de caractère (contenant du HTML), et (3) l'input est le résultat d'un get().

Les classes de HTML sont :

HTML : :Parser : cette classe est purement abstraite. Elle définit les méthodes objet suivantes ($h est un objet de la classe) :

- $h->new() : nouveau parseur,
- $h->parse($string) : analyse $string comme l'élément courant de l'objet
- $h->parse_file($file) : analyse directement le texte depuis $file
- $h->eof() : signale la fin du document HTML
- $h->strict_comment([$bool]) : si $bool=1, impose la syntaxe stricte pour les commentaires HTML
- $h->strict_names([$bool]) : si $bool=1, impose la syntaxe stricte pour les (quelques) noms d'attributs
- $h->boolean_attribute_value($val) : affecte $val aux attributs booléens à l'intérieur des balises de départs HTML. (???)
- $h->handler : sert à affecter une fonction de manipulation d'event (text, start, end, declaration, comment, process, default)

HTML : :LinkExtor : Cette classe, sous-classe de HTML : :Parser (et donc accédant à toutes les méthodes de sa superclasse), introduit deux nouvelles méthodes pour l'extraction de liens dans un document HTML ($h est un objet de la classe) :

- $h->new([\&callback [, $base]]) : crée un nouvel objet de la classe. Les arguments optionnels sont décrits plus loin
- $h->links() : renvoie la liste de tous les liens, chacun ayant la forme d'une référence à une liste anonyme : [$tag, $attr => $url1, $attr2 => $url2,...]

HTML : :Element permet de représenter des éléments dans un arbre d'analyse HTML ($h est un objet de la classe) :

- $h->new()
- $h->attr('attr') ou attr('attr', 'value') : renvoie la valeur d'un attribut, ou réaffecte un attribut (dans ce cas, l'ancienne valeur est retournée)
- $h->tag() ou tag('nom') : retourne (ou réaffecte) le nom de la balise (ou de l'identificateur générique) de l'élément.
- $h->parent() ou parent($new_parent) : renvoie ou reaffecte le parent de l'élément. Le parent, s'il est définit est également un objet de la classe Element.
- $h->content_list() : renvoie la liste représentant le contenu de l'élément
- $h->pos() ou pos($element) : renvoie la position d'un élément dans l'arborescence
- $h->all_attr() : renvoie le nom/valeur de tous les attributs sous forme clé,val
- $h->all_attr_names() : renvoie le nom de tous les attributs
- $h->push_content($elem, ...) : ajoute des éléments à la fin de la structure
- ...(autres méthodes de modification de structure)

HTML : :TokeParser : autre interface pour l'analyse syntaxique ($h est un objet de la classe) :

- $h->new($fichier)
- $h->get_token() : renvoie l'élément suivant dans l'arbre, sous forme de référence à un tableau, de la forme :
  ```
  ["S",  $tag, $attr, $attrseq, $text]
  ["E",  $tag, $text]
  ["T",  $text, $is\_data]
  ["C",  $text]
  ["D",  $text]
  ["PI", $token0, $text]
  ```
- $h->get_tag([$tag, ...]) : renvoie le tag de début ou de fin suivant
- $h->get_text([$endtag, ...]) : renvoie le text rencontré à la position courante
- $h->get_trimmed_text([$endtag, ...]) : renvoie le text rencontré à la position courante de début ou de fin, en réduisant à un espace l'ensemble des espace, tabs, retour à la ligne, etc. rencontrés.

HTML : :TreeBuilder : analyseur qui construit un arbre d'analyse

HTML : :Entities : code/décode les diacritiques

Note au passage : il y a bien d'autres modules dans la distribution d'ActivePerl. Il y en a encore plus dans la CPAN (cf. cours d'introduction aux modules et librairies Perl). Je vous présente ceux que je connais pour les avoir « décryptés » et utilisés. A vous de mettre le nez dans les autres modules de la librairie HTML, histoire de voir si quelquechose vous semble intéressant. Par ailleurs, tout ça étant libre, les contenus sont susceptibles d'évoluer, on doit absolument se baser sur la documentation fournie par les différents auteurs pour décider, en fonction de son problème, quelle méthode utiliser.

Exemples d'applications

Dans les exercices ci-dessous, on peut supposer que le document HTML est :
- soit acquis (via un get), il s'agit alors d'une chaîne de caractères
- soit sous forme de fichier : fournir ici tous les fichiers nécessaires

On va :
- extraire tous les liens (ou des liens choisis) à partir d'un doc. HTML
- afficher le texte contenu dans un doc HTML
- décortiquer attribut par attribut les différents éléments d'un doc HTML (vous rappelez-vous de l'exercice dans le chapitre "Structures complexes")
- extraire le contenu des éléments d'un certain type, dans les doc. HTML
- extraire les éléments ayant une caractéristique graphique particulière (ex : en italique)
- corriger la syntaxe d'un doc. HTML

Qu'est ce que HTML ?

Un document HTML peut-être vu comme un arbre. La racine est le token enchâssant le plus périphérique (<HTML ...> ... </HTML>), les tokens de même niveau (c'est à dire les tokens T1 T2 tels que <T1> ne contient pas <T2> et </T1>précède <T2>) sont frères.

Exemple :

```
<html lang='en-US'>
    <head>
      <title>Stuff</title>
      <!-- I like Pie.
          Pie is good
      -->
      <!DOCTYPE foo>
```

```
      <meta name='author' content='Jojo'>
    </head>
    <body>
     <h1>I like potatoes!</h1>
     <?stuff foo?>
    </body>
</html>
```

Analyse arborescente d'un document HTML (HTML : :Element, HTML : :Tree)

Dans ces circonstances, le document HTML ci-dessus peut être vu comme une suite structurée d'éléments. Chaque élément est repérable par les parties suivantes :

tag (balise) ouvrant et fermant,
parent (sauf la racine),
contenu : liste des éléments fils ou liste réduite à un segment de texte
attribut1 : val1
attribut2 : val2
...
attributn : valn

Ce qui produit la représentation formelle suivante :

```
element #1:  \_tag: 'html'
             \_parent: none
             \_content: [element #2, element #5]
             lang: 'en-US'

element #2:  \_tag: 'head'
             \_parent: element #1
             \_content: [element #3, element #4]

element #3:  \_tag: 'title'
             \_parent: element #2
             \_content: [text segment "Stuff"]

element #4   \_tag: 'meta'
             \_parent: element #2
             \_content: none
             name: author
             content: Jojo

element #5   \_tag: 'body'
             \_parent: element #1
             \_content: [element #6]
```

```
element #6   \_tag: 'h1'
             \_parent: element #5
             \_content: [text segment "I like potatoes"]
```

C'est ce que permet le module HTML : :Element.

Analyse d'un document HTML sous forme de tokens (HTML : :TokeParser)

Cette deuxième façon d'analyser un document HTML est (à mon avis, je me trompe peut-être!!!!) plus intéressante si notre but est de rechercher (1) à extraire le texte seul d'un doc. HTML ou encore (2) d'extraire automatiquement et itérativement tous les liens référencés à partir d'un dopc. donné. Au cours de l'analyse syntaxique, les méthodes de ce module font apparaître plusieurs types de structures correspondant aux différents 'tokens' observables dans un doc. HTML :

```
["S",  $tag, $attr, $attrseq, $text] :
Exemple : <meta name='author' content='Jojo'>
[ "S", meta,
{ name => "author", content => "Jojo" },
[content,name],
<meta name='author' content='Jojo'>
]

["E",  $tag, $text]:
Exemple : </head>
[ "E",head,</head> ]

["T",  $text, $is\_data] :
Exemple : I like potatoes!
[ "T",I like potatoes! ]

["C",  $text] :
Exemple : <!-- I like Pie. Pie is good -->
[ "C",<!-- I like Pie. Pie is good --> ]

["D",  $text] :
Exemple : <!DOCTYPE foo>
[ "D",<!DOCTYPE foo> ]

["PI", $token0, $text]
Exemple : <?stuff foo?>
[ "PI",<?stuff foo?> ]
```