Проект по извлечению из HTML страницы значимого контента и изображений. Работает с применением библиотеки HtmlAgilityPack.

Описание проекта HtmlCleaner

Проект применяет библиотеку HtmlTree для построения дерева, и содержит логику по получению параметров приложения из настроек и консоли, вызов API HtmlTree для построения и анализа дерева и сохраняет полученные результаты в файлы.

В проекте применена библиотека NInject для снятия зависимости в проекте от конкретной реализации фабрики, и описания всех зависимостей в одном модуле.

Описание проекта HtmlCleanerTest

Unit-тест проект, предназначен для быстрого тестирования результатов построения во время разработки.

Описание библиотеки HtmlTree

Библиотека создана для разбора Html-кода в Html-дерево и проставления ключевых атрибутов узлам полученного дерева. На основе проставленных атрибутов принимается решение о значимости узлов деревьев и необходимости их отображения.

В пространстве имен Meta описаны интерфейсы, которые можно применять во внешних приложениях, использующих эту библиотеку.

Абстрактная фабрикой IHtmlNodeFactory, предназначена для создания дерева. Для создания дерева передаются настройки IHtmlTreeBuildSettings.

В проекте приведена реализация построения дерева на основе применения библиотеки HtmlAgilityPack.

В пространстве имен HtmlAgilityPackBased описан класс HtmlNodeFactory, который инжектится в проект.

Разбор Html в дерево.

Первым этапом выполняется разбор HTML в структуру HtmlAgilityPack.HtmlNode. Структура древовидная.

На основе данной структуры создается дерево из значимых узлов. Дерево создается обходом в глубину исходного дерева в методе ConstructTree класса HtmlNodeFactory. Обход в глубину реализован в цикле. Если есть необработанные дочерние узлы, заходим на обработку к ним, иначе поднимаемся к предку и продолжаем обработку. Во время построения дерева игнорируются узлы, название которых попадает в стоп-лист (такие как script). Выполняется очистка контента на основе списка регулярных выражений для очистки.

После построения дерева выполняется обработка узлов-изображений. Если src-аттрибут узла img пуст, то узел пропускается. Иначе - URL изображения добавляется в список связанного контента узла. В зависимости от BaseURL, на основе которого выполняется построение дерева, URL преобразуется в абсолютный URL.

Следующим этап проходит обработка ссылок. Выполняется преобразование ссылки в абсолютную ссылку и проверка на значимость URL. Незначимыми URL считаются содержащие "javascript" в начале, "#", "mailto", или ссылки, оканчивающиеся на характерные для изображений расширения. Если ссылка значимая, в узел ссылки добавляется лист, содержащий в контенте [URL] и отмечается флагом NodeType.Hyperlink, иначе - узел пропускается. Затем увеличивается количество ссылок в родительских узлах.

Затем выполняется расчет ключевых показателей узлов дерева. Выполняется это двумя методами - один выполняется до входа в потомков, другой после выхода из них. До входа в потомков расчитывается уровень узла, после входа в потомков расчитывается количество предложений в узле. В узлах количество предложений зависит от количества предложений в его потомках, в листах это количество предложений расчитывается на основе превышения количества слов заданного значения и количества символа разделителя предложения - ". ", ".\n" и т.д.

Следующим этапом маркируются заголовки дерева (h1-h9). Наличие внутри узла заголовка проставляется во всех родительских узлах.

Далее выполняется фильтрация контента. Если узел-лист, очищается контент от повторных пробелов и сбросов строки. Если гиперссылка, поднимается до узла - предка, у которого количество потомков превышает 1 и количество заголовков нулевое. Если такой узел найден, и в нем количестов листов меньше количества ссылок, то такой узел считаем навигационным и не важным для отображения и пропускам. Если узел - блок (div или p по базовым настройкам), не заголовок, количесто предложений в нем равно 0, и количество заголовков также равно нулю, то такой блок пропускам.

Следующим этапом обрабатываются весь связанный контент (изображения). Т.к. мы могли в предыдущих этапах скрыть значимые изображения, или наоборот добавить их, поднимаем в предков весь связанный контент.

Далее все изображения внедряются в текст узлов по следующему алгоритму: Если узел-лист и содержит изображения, то изображения включаются в текст контента. Если узел-скрытый (отмеченный для пропуска), но его прямой родитель является отображаемым узлом, то в контент узла включаются ссылки на изображения.

Следующим этапом выполняется включение контента из дочерних узлов в родительские. По необходимости добавляются пробелы, между узлами соседями, сбросы строки. Если узел - заголовок, то в текст включается описатель заголовка.

Затем выполняется окончательная очистка контента в начале и в конце от пробелов и пропусков строк и на этом дерево считается построенным.

Обработка дерева

В полученном дереве выполняется поиск первого попавшегося узла-заголовка. Этот узел считается точкой отсчета. От него выполяется подъем к родителям и анализ каждого родительского узла на соотношение количества предложений от максимального количества предложений корневого узла. Если соотношение соответствует настройкам (по умолчанию задано 0.3, т.е. в узле должно быть "количество предложений в узле"<=0.3*"количество предложений в корне".), то узел считается узлом для отображения.

Весь контент до первого заголовка игнорируется.

Выполняется сохранение связанного с выбранным узлом контента и проставляются флаги успешности загрузки. Ссылки на не загруженный контент удаляются из формируемого для отображения текста, длинные GUID ссылки заменяются на короткие ссылки в непрерывной последовательности [1..количество загруженных изображений], соответствующие названиям загруженных файлов.

Далее выполняется форматирование контента, с правилом наличия в строке не более 80 (меняется настройками) символов. Если слово превышает в длину 80 символов, оно разносится на несколько строк. Словом называется непрерывная последовательность символов, несодержащая пробела.

И в итоге этот контент сохраняется в текстовый файл.

Пути для улучшения

1) Улучшить алгоритм поиска начального узла (с которого будет выборка значимого контента).
1.1) Обработать случай, когда на странице нет заголовков, в таком случае нужно описать алгоритм выбора оптимального начального узла.

2) Улучшить алгоритм исключения узлов с навигацией. Узлы с навигацией определяются по количеству ссылок и предложений. Возможно есть более "чистый" способ отсечения навигации.

3) Найти алгоритм исключения подвала страницы. Подвал страницы (зона где копирайта) часто попадает под целевой контент. Возможно его исключать по наличию копирайты.

4) Возможна ситуация, когда первоначальный заголовок находится в шапке страницы, в промежутке между ним и значимым контентом находится некоторый контент, например навигация. В таком случае, нужно установить критерии, по которым принимать данный заголовок незначимым заголовком, и выбирать следующий заголовок для поиска стартового узла (с которого будет проводится поиск значимого контента). Правильная выборка стартового узла дает более чистый результат, чем выбор узла, содержащего практически весь контент страницы и определение незначимых узлов внутри него.

5) Это должно web/desktop приложения, в котором должна быть форма для настройки параметров и возможность превью результата.