# DOMDocument::loadHTMLFile

(PHP 5)

### Описание [¶](http://www.php.net/manual/ru/domdocument.loadhtmlfile.php#refsect1-domdocument.loadhtmlfile-description)

public bool **DOMDocument::loadHTMLFile** ( string $filename [, int $options = 0 ] )

Функция разбирает HTML документ из файла *filename*. В отличие от загрузки XML, HTML документ не обязан быть правильно построенным.

### Список параметров [¶](http://www.php.net/manual/ru/domdocument.loadhtmlfile.php#refsect1-domdocument.loadhtmlfile-parameters)

*filename*

Путь к HTML файлу.

*options*

Начиная с версии PHP 5.1.0 и Libxml 2.6.0, можно также использовать параметр *options* для передачи [дополнительных параметров Libxml](http://www.php.net/manual/ru/libxml.constants.php).

### Возвращаемые значения [¶](http://www.php.net/manual/ru/domdocument.loadhtmlfile.php#refsect1-domdocument.loadhtmlfile-returnvalues)

Возвращает **TRUE** в случае успешного завершения или **FALSE** в случае возникновения ошибки. Если вызвана статически, возвращает объект класса [DOMDocument](http://www.php.net/manual/ru/class.domdocument.php) или **FALSE** в случае возникновения ошибки.

### Ошибки [¶](http://www.php.net/manual/ru/domdocument.loadhtmlfile.php#refsect1-domdocument.loadhtmlfile-errors)

Если через аргумент *filename* передана пустая строка или файл ничего не содержит, будет сгенерировано предупреждение. Это предупреждение генерируется не libxml, поэтому оно не может быть обработано [обработчиками ошибок библиотеки libxml](http://www.php.net/manual/ru/function.libxml-use-internal-errors.php).

Этот метод может быть вызван статически, но при этом будет сгенерирована ошибка уровня **E\_STRICT**.

Несмотря на то, что некорректный HTML обычно успешно загружается, данная функция может генерировать ошибки уровня **E\_WARNING** при обнаружении плохой разметки. Для обработки данных ошибок можно воспользоваться [функциями обработки ошибок libxml](http://www.php.net/manual/ru/function.libxml-use-internal-errors.php).

### Примеры [¶](http://www.php.net/manual/ru/domdocument.loadhtmlfile.php#refsect1-domdocument.loadhtmlfile-examples)

**Пример #1 Создание документа**

<?php  
$doc = new DOMDocument();  
$doc->loadHTMLFile("filename.html");  
echo $doc->saveHTML();  
?>

Те, кто хоть раз писал парсер, знает, что не стоит этого делать с помощью регулярных выражений. Проиллюстрировать это утверждение поможет следующий пример.

 Возьмем HTML код:

|  |
| --- |
| <div><a href="<http://xdan.ru>"><div>Сайт по программированию парсеров</div><div> и многое другое</div></a></div> |

К примеру, из него нам нужно получить описание и url сайта. Если брать исключительно этот кусок кода, то все решается достаточно просто:

|  |
| --- |
| $html = '<div><a href="<http://xdan.ru>"><div>Сайт по программированию парсеров</div><div> и многое другое</div></a></div>';  preg\_match('#<div><a href="([^"]+)"><div>([^<]+)</div><div>([^<]+)</div></a></div>#U',$html,$list);  echo 'url:'.$list[1].',title:'.$list[2].$list[3]; // выведет url:<http://xdan.ru>,title:Сайт по программированию парсеров и многое другое |

Проблемы начинаются тогда, когда описание сайта заполняют пользователи, и оно не имеет   определенного шаблона.

|  |
| --- |
| <div><a href=”<http://xdan.ru>”><div>Сайт по <b>программированию</b> парсеров</div><div> и многое <div> многое </div> другое </div></a></div> |

Такой код регулярному выражению не по зубам.

Обычно, в вузах на этот случай учат писать конечный автомат. Суть его в том, что мы перебираем, посимвольно, весь html текст, находим начало тега, и строим дерево документа. Так называемое DOM (Document Object Model)

Сейчас, писать такое самому  нет необходимости.

В php, начиная с  версии 5, есть встроенные методы работы с деревом документа (класс DOMDocument), но основан он на XML парсере.

А HTML и XML это хоть и очень похожие, но в тоже время абсолютно разные технологии.

К примеру, непременное требование к XML это закрытые теги и отсутствие ошибок.

Отсюда вытекает условие: ошибок в html, который мы парсим с помощью нативных средств php,  быть не должно.

К сожалению, на сайтах донорах, ошибки не редки, а значит этот метод отпадает.

Для корректного разбора таких сайтов, на помощь придут php библиотеки PHPQuery, Simple HTML DOM, Zend DOM Query, Nokogiri .

Некоторые из них, после небольших манипуляций скармливают html  тому же DOMDocument. Мы не будем их рассматривать.

В этой статье я расскажу про SimpleHTMLDOM. Этой библиотекой я пользуюсь уже несколько лет, и она меня еще ни разу не подводила.

Скачиваем последнюю версию [здесь](http://sourceforge.net/projects/simplehtmldom/files/simplehtmldom/).

Пусть Вас не смущает то, что она не обновлялась с 2008 года, то, что она умеет, полностью покроет Ваши нужды в разборе html текстов.

В архиве, который вы скачали, две папки (примеры работы и документация) и файл simple\_html\_dom.php.

simple\_html\_dom.php это и есть вся библиотека, больше ничего для работы не потребуется. Кидаем этот файл в папку с проектом и в своем скрипте просто подгружаем его.

|  |
| --- |
| include 'simple\_html\_dom.php'; |

Кроме документации, которую вы скачали с архивом, доступна еще online версия, ее  вы найдете [здесь](http://simplehtmldom.sourceforge.net/manual.htm)

Файл подключен и готов к работе.

Для того, чтобы начать разбирать HTML, его сперва нужно получить. Обычно, я делаю это при помощи библиотеки CURL.

В simplehtmldom есть методы для удаленной загрузки страниц.  После подключения файла библиотеки, нам доступны 2 функции для обработки HTML строк.

str\_get\_htm(str) и file\_get\_html(url)

Они  делают одно и тоже, преобразуют HTML текст в DOM дерево, различаются лишь источники.

str\_get\_htm – на вход получает обычную строку, т.е. если вы получили HTML прибегнув к curl, или file\_get\_contents то вы просто передаете полученный текст этой функции.

|  |
| --- |
| $html = str\_get\_html('<html><body>Привет!</body></html>'); |

file\_get\_html – сама умеет загружать данные с удаленного URL или из локального файла

|  |
| --- |
| $html = file\_get\_html('<http://www.yandex.ru/>'); |

или

|  |
| --- |
| $html = file\_get\_html('data/test.htm'); |

К сожалению,  file\_get\_html загружает страницы обычным file\_get\_contents. Это значит если хостер, выставил в php.ini allow\_url\_fopen = false (т.е. запретил удаленно открывать файлы), то загрузить что-то удаленно, не получится. Да и серьезные веб сайты таким способом парсить не стоит, лучше использовать CURL с поддержкой proxy и ssl. Однако для наших опытов, вполне хватит и file\_get\_html.

|  |
| --- |
| $html = file\_get\_html('<http://www.yandex.ru/>'); |

в результате, в  переменной $html будет объект типа simple\_html\_dom.

При больших объемах данных, в библиотеке происходит утечка памяти. Поэтому после окончания одного цикла надо ее чистить.

Делает это метод clear.

К примеру грузим 5 раз сайт www.yandex.ru с разными поисковыми запросами

|  |
| --- |
| $k = 5;  while($k>0){   $html = file\_get\_html('<http://yandex.ru/yandsearch?text=hi>'.$k.'&lr=11114'); // загружаем данные   // как-то их обрабатываем   $html->clear(); // подчищаем за собой   unset($html);   $k--;  } |

Эти две строчки $html->clear();  и unset($html);  лучше писать сразу же после того, как  Вы создали объект. Иначе забудете, и скрипт отвалится, забив всю память.

После того, как html текст упакован в объект, можно приступать непосредственно к поиску нужных элементов.

Большинство поисковых функций выполняет метод find(selector, [index]).  Если второй аргумент не задан, метод возвращает массив элементов. Если же задан то элемент этого массива с индексом index.

Пример: скачаем главную страницу моего блога, и выведем все ссылки, которые встретим на своем пути.

|  |
| --- |
| require\_once 'simple\_html\_dom.php';  $data = file\_get\_html('<http://xdan.ru>');  if($data->innertext!='' and count($data->find('a'))){      foreach($data->find('a') as $a){          echo '<a href="<http://xdan.ru/>'.$a->href.'">'.$a->plaintext.'</a></br>';      }  } |

В примере, в качестве селектора я воспользовался названием тега <a>. Но можно использовать и другие CSS селекторы. Элемент на странице можно найти по его атрибутам. В первую очередь, это название тега, id и class. Также могут быть использованы и второстепенные атрибуты, к примеру, href ссылки или width картинки. Если и этих атрибутов нет, то не грех воспользоваться и регулярными выражениями.

Поиск по названию тега вы уже видели

|  |
| --- |
| $html->find('div') |

поиск по id

|  |
| --- |
| $html->find('#preview') |

поиск по классу

|  |
| --- |
| $html->find('.myclass') |

или комбинированный вариант

|  |
| --- |
| $html->find('#preview div.myclass') |

в данном случае, сначала найдется элемент с id= preview затем в нем найдутся все теги div, и уже среди них фильтруются те у которых class=”myclass”

Если метод find ничего не нашел и index не задан, то он возвращает пустой массив. Если же index задан, то метод возвращает null.

Поэтому верным решением будет проверить

|  |
| --- |
| if(count($html->find('#preview div.myclass')))      foreach($html->find('#preview div.myclass') as $div)          echo $div->innertext; |

Поиск по наличию атрибута

|  |
| --- |
| $html->find(' img [width]'); // найдет нам все изображения у которых задан атрибут ширина |

или более конкретный поиск по значению атрибута

|  |
| --- |
| $ret = $html->find('img[width=400px]');// найдет все изображения, у которых задана ширина равная 400px |

Такая нотация позволяет искать по двум и более смежным классам

|  |
| --- |
| $ret = $html->find('img[class=active myclass]');//<img class="active myclass"/> |

Поиск нескольких тегов

|  |
| --- |
| $html->find('a, img, br,span'); |

Поиск вложенных тегов

|  |
| --- |
| $es = $html->find('ul.myclass li');// найдет все li который является потомком ul(возможно и не прямым)  $es = $html->find('div.myclass li');// найдет все li в  div.myclass |

У каждого найденного элемента также есть метод find

|  |
| --- |
| $html->find('div.myclass li');//найдет все div.myclass а потом все li лежащие в них |

если нам нужно найти все li только первого div’а то мы можем написать так

|  |
| --- |
| $html->find('div.myclass',0)->find('li'); |

Поиск по значению атрибута не ограничивается только равенством. Вот доступные условия

[атрибут] – проверяет есть ли у элемента данный атрибут

[атрибут=величина] - проверяет, есть ли у элемента данный атрибут и равно ли его значение величине.( div[class=myclass] – найдет все div’ы у которых class равен myclass)

[атрибут!=величина] - проверяет, есть ли у элемента данный атрибут и не равно ли его значение величине.( div[class!=myclassok] – найдет все div’ы у которых class не равен myclassok)

[атрибут^=величина] - проверяет, есть ли у элемента данный атрибут и начинается ли его значение с величины ( div[class^=my] – найдет все div’ы у которых class начинается с my, к примеру myclass и myclassok)

 [атрибут$=величина] - проверяет, есть ли у элемента данный атрибут и заканчивается ли его значение величиной( div[class$=ok] – найдет все div’ы у которых class заканчивается на ok, к примеру myclassok, yok, okно не oki)

 [атрибут\*=величина] - проверяет, есть ли у элемента данный атрибут и содержит ли его значение в себе величину, в любом месте(div[class\*=sok] – найдет все div’ы у которых class содержит sok, к примеру myclassok, ysoki, sok)

Обычный текст можно искать как тег text

|  |  |
| --- | --- |
| 1 | $es = $html->find('text'); // найдет все текстовые блоки в html |

Комментарии находим по тегу comment

|  |  |
| --- | --- |
| 1 | $es = $html->find('comment'); |

Каждый найденный элемент и сам $html имеют 5 полей

|  |
| --- |
| $html = str\_get\_html("<div>foo <b>bar</b></div>");  echo $html; // выведет <div>foo <b>bar</b></div>;  $e = $html->find("div", 0);  echo $e->tag; // Вернет: "div"  echo $e->outertext; // Вернет: <div>foo <b>bar</b></div>  echo $e->innertext; // Вернет: foo <b>bar</b>  echo $e->plaintext; // Вернет: foo bar |

$e->tag            Читает или записывает имя тега элемента.

$e->outertext   Читает или записывает весь HTML элемента, включая его самого.

$e->innertext   Читает или записывает внутренний HTML элемента

$e->plaintext    Читает или записывает простой текст элемента, это эквивалентно функции strip\_tags($e->innertext). Хотя поле доступно для записи, запись в него ничего не даст, и исходный html не изменит

|  |
| --- |
| $html = str\_get\_html("<div>foo <b>bar</b></div");  $div = $html->find('div',0);  $div->plaintext = 'gooo';  echo $div->innertext; // вернет <div>foo <b>bar</b></div> |

Как  Вы могли догадаться, для удаления ненужного элемента из HTML можно затереть его поле  outertext

|  |
| --- |
| $html = str\_get\_html("<div>foo <b>bar</b></div");  $b = $html->find('b',0);  $b->outertext = '';  echo $html->innertext; // вернет <div>foo</div> |

Тут следует помнить, что хоть элемент и не виден в html, из дерева DOM он никуда не делся

|  |
| --- |
| $html = str\_get\_html("<div>foo <b>bar</b></div");  $b = $html->find('b',0);  $b->outertext = '';  echo $html->innertext; // вернет <div>foo</div>, элемент удален из HTML  // но  echo count($html->find('b')); // вернет 1, в дерево элемент присутствует |

при желании  мы даже можем вернуть элемент на место

|  |  |
| --- | --- |
| 1  2 | $b->outertext = '<span>bar</span>';  echo $html->innertext;// вернет <div>foo<span>bar</span></div> |

Для более эффективной навигации по дереву документа доступны методы

$e->children ( [int $index] )     Возвращает объект N-го прямого потомка, если индекс установлен, в противном случае возвращает массив всех дочерних элементов

$e->parent()    Возвращает родительский элемент.

$e->first\_child() Возвращает первый дочерний элемент, или null, если ничего не найдено

$e->last\_child() Возвращает последний дочерний элемент, или null, если ничего не найдено

$e->next\_sibling() Возвращает следующий родственный элемент, или null, если ничего не найдено

$e->prev\_sibling()       Возвращает предыдущий родственный элемент, или null, если ничего не найдено

пример

|  |
| --- |
| $html ="<div>    <b>bar</b>    <b>foo</b>    <span>arg</span>    <div>      <b>tor</b>    </div>  </div>"; |

Все дочерние элементы разные, как-то подобрать к ним селектор проблематично. Поэтому воспользуемся описанными методами.

|  |
| --- |
| $html  = str\_get\_html($html);  $div = $html->find('div',0);  $i = 0;  while($item = $div->children($i++)){      echo $item->innertext;  } |

либо так

|  |
| --- |
| $item = $div->children(0);  echo $item->innertext;  while($item = $item -> next\_sibling()){     echo $item->innertext;  } |

Данные методы полезны при разборе таблиц, элементы которых, как правило, структурированы, но не имеют идентифицирующих атрибутов.

Ну и последняя фишка это вызов callback функции на найденный элемент

|  |
| --- |
| function my\_callback($element) {         if ($element->tag=='span')                  $element->outertext = '<b>'.$element->innertext. '</b>';// заменим все span элементы на b  }  $html  = str\_get\_html('<span>bar</span><span>pole</span><span>sushi</span><a>okno</a>');  // Регистрация функции обратного вызова с ее именем  $html->set\_callback('my\_callback');// вызов функции произойдет при конвертации объекта в строку  echo $html; // на самом деле, при этом вызывается магический метод \_\_toString, он и запускает наши калбяки |

На экране мы увидим

|  |  |
| --- | --- |
| 1 | <b>bar</b><b>pole</b><b>sushi</b><a>okno</a> |

Доступ к атрибутам элементов осуществляется напрямую

|  |
| --- |
| foreach($html->find('img') as $img) echo $img->src;  //или  echo $html->find('img',0)->src; |

Хватит теории, перейдем к практике

Загрузим n фотографий из поисковой выдачи Yandex Картинок. <http://images.yandex.ru/>

|  |
| --- |
| require\_once 'simple\_html\_dom.php';  // поисковый URL  $url = '<http://images.yandex.ru/yandsearch?text=>'.urlencode('Джессика Альба').'&rpt=image';  $n = 2;  // загружаем данный URL  $data = file\_get\_html($url);  // очищаем страницу от лишних данных, это не обязательно, но когда HTML сильно захламлен бывает удобно почистить его, для дальнейшего анализа  foreach($data->find('script,link,comment') as $tmp)$tmp->outertext = '';  // находим все изображения на странице  if(count($data->find('div.b-image img'))){      $i = 1;      foreach($data->find('div.b-image img') as $img){          // выводим на экран изображение          echo '<img src="'.$img->src.'"/>';          // и скачиваем его в файл          file\_put\_contents('data/'.($i++).'.jpg',file\_get\_contents($img->src));          if($i>$n)break; // выходим из цикла если скачали достаточно фотографий      }  }  $data->clear();// подчищаем за собой  unset($data); |

Как быть если нам нужно больше фото, чем лежит на одной странице?

Ответ прост: Код, приведенный выше, заключается в функцию, в html помимо фото находим еще  и URLвсех страниц, и рекурсивно вызываем данную функцию для этих страниц.

|  |
| --- |
| require\_once 'simple\_html\_dom.php';  function getYandexImages($url,$findpages = true){      static $i = 1;      $n = 200;      // загружаем данный URL      $data = file\_get\_html($url);      // очищаем страницу от лишних данных, это не обязательно, но когда HTML сильно захламлен бывает удобно почистить его, для дальнейшего анализа      foreach($data->find('script,link,comment') as $tmp)$tmp->outertext = '';      // находим URL страниц только для первого вызова функции      if( $findpages and count($data->find('div.b-pager\_\_pages a'))){          foreach($data->find('div.b-pager\_\_pages a') as $a){              // довольно распространенный случай - локальный URL. Поэтому иногда url надо дополнять до полного              if( !preg\_match('#^[http://](NULL)#',$a->href) )$a->href = '<http://images.yandex.ru>'.$a->href;              // и еще дна тонкость, &amp; надо заменять на &              $a->href = str\_replace('&amp;','&',$a->href);              // вызываем функцию для каждой страницы              getYandexImages($a->href,false);          }      }      // находим все изображения на странице      if(count($data->find('div.b-image img'))){          foreach($data->find('div.b-image img') as $img){              // выводим на экран изображение              echo '<img src="'.$img->src.'"/>';              // и скачиваем его в файл              file\_put\_contents('data/'.($i++).'.jpg',file\_get\_contents($img->src));              if($i>$n)exit; // завершаем работу если скачали достаточно фотографий          }      }      $data->clear();// подчищаем за собой      unset($data);  }  // поисковый URL  $url = '<http://images.yandex.ru/yandsearch?text=>'.urlencode('Джессика Альба').'&rpt=image';  getYandexImages($url); |

Все хорошо, 200 картинок лежат в папке data. Но их размер слишком мал.

Поэтому завершающим аккордом нашей практики будет загрузка увеличенной фотографии.

Для этого определим еще одну функцию

|  |
| --- |
| function getBigImage($url){      $data = @file\_get\_contents($url);      if(trim($data)=='')return false; // бывает что сайт недоступен, его фото мы не грузим      $data = str\_get\_html($data);      // находим фото      if( count($data->find('#i-main-pic')) ){          $dataimg = @file\_get\_contents($data->find('#i-main-pic',0)->src); // собачка нужна в если сервер нам вернул 404, это выозвет Warning:, поэтому экранируем ошибки          if(trim($dataimg)=='')return false; // фото не доступно, его не грузим          file\_put\_contents( 'data/'.md5($url).'.jpg', $dataimg ); // сохрпаняем в файл      }      $data->clear();// подчищаем за собой      unset($data);  } |

и слегка поправим getYandexImages

|  |
| --- |
| function getYandexImages($url,$findpages = true){      global $i,$n;      // загружаем данный URL      $data = @file\_get\_contents($url);      $data = str\_get\_html($data);      // очищаем страницу от лишних данных, это не обязательно, но когда HTML сильно захламлен бывает удобно почистить его, для дальнейшего анализа      foreach($data->find('script,link,comment') as $tmp)$tmp->outertext = '';      // находим URL страниц только для первого вызова функции      if( $findpages and count($data->find('div.b-pager\_\_pages a'))){          foreach($data->find('div.b-pager\_\_pages a') as $a){              // довольно распространенный случай - локальный URL. Поэтому иногда url надо дополнять до полного              if( !preg\_match('#^[http://](NULL)#',$a->href) )$a->href = '<http://images.yandex.ru>'.$a->href;              // и еще дна тонкость, &amp; надо заменять на &              $a->href = str\_replace('&amp;','&',$a->href);              // вызываем функцию для каждой страницы              getYandexImages($a->href,false);          }      }      // находим все изображения на странице      if(count($data->find('div.b-image img'))){          foreach($data->find('div.b-image a') as $a){              if( !preg\_match('#^[http://](NULL)#',$a->href) )$a->href = '<http://images.yandex.ru>'.$a->href;              $a->href = str\_replace('&amp;','&',$a->href);              getBigImage($a->href);              if($i++>=$n)exit; // завершаем работу если скачали достаточно фотографий              echo '<script>document.getElementById("counter").innerHTML = "Загружено: '.$i.' из '.$n.' фото";</script>';              flush();          }      }      $data->clear();// подчищаем за собой      unset($data);  }  // поисковый URL  $i = 1;  $n = 20; // будем грабить 20 картинок  $url = '<http://images.yandex.ru/yandsearch?text=>'.urlencode('Джессика Альба').'&rpt=image';  getYandexImages($url); |

Вот и все, наслаждаемся фото великолепной Джессики Альбы. Надеюсь меня простит Яндекс, ведь по сути фото грабится не с их серверов, а с прямиком с сайтов, где они лежат.

Кроме того это всего лишь демонстрация работы. Думаю никому в здравом уме, не придет в голову парсить Яндекс с помощью file\_get\_content. Данную библиотеку можно применять и в мирном программировании. К примеру в качестве шаблонизатора для CMS. Почему нет, с хорошим кешированием будет очень удобная штука.

 При больших объемах сайтов доноров,  неплохо бы [разбить все на потоки](http://xdan.ru/Multi-threaded-parsers.html).

А используя описанный мой [скрипт сортировки изображений по цвету](http://xdan.ru/Kak-opredelit-preobladayuschiy-cvet-v-izobrajenii.html), можно собрать неплохую отсортированную базу, фотографий знаменитостей.

И, как всегда, выкладываю все [исходники](http://xdan.ru/down/file/10)

**UPD: Благодаря пользователю Диме, выяснилось, что библиотека обновилась 2011-07-14 и имеет версию 1.5. Нововведений не много, однако самый главный баг, с которым сталкивались все, кто пользовался библиотекой, а именно утечка памяти, устранен. На то она и версия 1.5, а не 2.0. Радует, что проект все же поддерживается и развивается. Ждем новых релизов.**

C#

HtmlWeb htmlWeb = new HtmlWeb();  
HtmlDocument doc = htmlWeb.Load(«http://www.bbc.co.uk/»);

HtmlNodeCollection links =  
doc.DocumentNode.SelectNodes(«//a[@href]«);

foreach (HtmlNode link in links)  
{  
Response.Write(link.Attributes["href"].Value + »  
«);  
}

**Беглый осмотр Html Agility Pack**

Справку по библиотеке можно скачать на странице проекта. Функционал на самом деле очень радует.   
Всего нам доступно двадцать основных классов:  
  
[](http://habrastorage.org/storage/habraeffect/4d/96/4d96e96a153dced2af6f6b6f97af9693.png)  
  
Названия методов соответствуют [интерфейсам DOM](http://www.w3.org/TR/DOM-Level-2-Core/core.html) ([замечание](http://habrahabr.ru/blogs/net/112325/#comment_3594351) [k12th](http://habrahabr.ru/users/k12th/)) + плюшки: GetElementbyId(), CreateAttribute(), CreateElement() и т.д., так что работать будет особенно удобно, если приходилось сталкиваться с JavaScript  
Похоже, что html все же перегоняется в Xml, а HtmlDocument и др. классы это обертка, ну и ничего страшного в этом, ввиду этого доступны такие возможности как:

* Linq to Objects (via LINQ to Xml)
* XPATH
* XSLT

**Парсим хабр!**

Вакансии на хабре представлены в виде таблицы, в строках дана информация о требуемой специальности и зарплате, но так как нам нужна информация об образовании, то придется переходить на страницу вакансии и разбирать ее.  
Итак, начнем, нам нужна таблица, чтобы вытащить оттуда ссылки и инфу о позиции с зарплатой:

1. static void GetJobLinks(HtmlDocument html)
2. {
3. var trNodes = html.GetElementbyId(«job-items»).ChildNodes.Where(x => x.Name == «tr»);
5. foreach (var item in trNodes)
6. {
7. var tdNodes = item.ChildNodes.Where(x => x.Name == «td»).ToArray();
8. if (tdNodes.Count() != 0)
9. {
10. var location = tdNodes[2].ChildNodes.Where(x => x.Name == «a»).ToArray();
12. jobList.Add(new HabraJob()
13. {
14. Url = tdNodes[0].ChildNodes.First().Attributes[«href»].Value,
15. Title = tdNodes[0].FirstChild.InnerText,
16. Price = tdNodes[1].FirstChild.InnerText,
17. Country = location[0].InnerText,
18. Region = location[2].InnerText,
19. City = location[2].InnerText
20. });
21. }
23. }
25. }

А после осталось пройти по каждой ссылке и вытащить инфу об образовании и заодно еще и занятость — здесь есть небольшая проблема в том, что если таблица с ссылками на вакансию лежала в div-е с известным id, то информация о вакансия лежит в таблице без всяких id, поэтому пришлось немножко поизвращаться:

1. static void GetFullInfo(HabraJob job)
2. {
3. HtmlDocument html = new HtmlDocument();
4. html.LoadHtml(wClient.DownloadString(job.Url));
5. // html.LoadHtml(GetHtmlString(job.Url));
7. // так делать нельзя :-(
8. var table = html.GetElementbyId(«main-content»).ChildNodes[1].ChildNodes[9].ChildNodes[1].ChildNodes[2].ChildNodes[1].ChildNodes[3].ChildNodes.Where(x => x.Name == «tr»).ToArray();
10. foreach (var tr in table)
11. {
12. string category = tr.ChildNodes.FindFirst(«th»).InnerText;
14. switch (category)
15. {
16. case «Компания»:
17. job.Company = tr.ChildNodes.FindFirst(«td»).FirstChild.InnerText;
18. break;
19. case «Образование:»:
20. job.Education = HabraJob.ParseEducation(tr.ChildNodes.FindFirst(«td»).InnerText);
21. break;
22. case «Занятость:»:
23. job.Employment = HabraJob.ParseEmployment(tr.ChildNodes.FindFirst(«td»).InnerText);
24. break;
25. default:
26. continue;
27. }
28. }
29. }

**Результаты**

Ну а дальше, сохраняем результаты в XML и смотрим в Excel-e, что же получилось… и видим, что ничего хорошего не получилось, потому что большинство компаний либо не указывают размер зарплаты, либо не указывают информацию об образовании (забывают, указывают в теле вакансии, или действительно неважно), либо не указывают все сразу.  
Кому интересно, вот результаты в [xlsx](http://rghost.ru/4031568) и [xml](http://rghost.ru/4031572), а [здесь](http://pastebin.com/RbmAD8VM) исходник

**P.S.**

При парсинге возникла такая проблема — страницы скачивались очень медленно. Поэтому я сначала попробовал WebClient, а потом WebRequest, но разницы не было. Поиск в гугле указал на то, что следует явно отключать Proxy в коде, и тогда все будет хорошо, однако это тоже не помогло.

## Definition and Usage

The file\_get\_contents() reads a file into a string.

This function is the preferred way to read the contents of a file into a string. Because it will use memory mapping techniques, if this is supported by the server, to enhance performance.

## Syntax

file\_get\_contents(path,include\_path,context,start,max\_length)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| path | Required. Specifies the file to read |
| include\_path | Optional. Set this parameter to '1' if you want to search for the file in the include\_path (in php.ini) as well |
| context | Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream. Can be skipped by using NULL. |
| start | Optional. Specifies where in the file to start reading. This parameter was added in PHP 5.1 |
| max\_length | Optional. Specifies how many bytes to read. This parameter was added in PHP 5.1 |

## Tips and Notes

**Tip:** This function is binary-safe (meaning that both binary data, like images, and character data can be written with this function).

## Example

<?php  
echo file\_get\_contents("test.txt");  
?>