

Руководство разработчика по использованию веб-типографа Студии Муравьева

Класс «Jare_Typograph», emuravjev.ru/works/tg/

Версия 2.0.0, программист Артур Русаков

Содержание

1. Начало работы	3
1.1. Вводная часть	3
1.2. Инициализации	3
1.2.1. Быстрая	3
1.2.2. Обычная	3
1.2.2.1. Ключевые методы	3
1.2.2.1.1. getBaseTofsNames()	3
1.2.2.1.2. getTof() и setTof()	3
1.2.2.1.3. parse()	4
2. Тофы	5
2.1. Обзор	5
2.2. Ключевые методы	5
2.2.1. getBaseParam() и setBaseParam()	5
2.2.2. disableParsing() и isDisabledParsing()	5
2.2.3. disableBaseParam()	5
2.3. Продвинутое использование	6
2.3.1. Создание собственных тофов	6
3. Параметры тофов	8
3.1. Обзор	8
3.2. Ключевые методы	8
3.2.1. getOption() и setOption()	8
3.2.2. reset()	9
3.2.3. disable()	9
4. Инструментарий	10
4.1. Обзор	10
4.2. clearSpecialChars()	10
4.3. removeHtmlTags()	10
4.4. Безопасные блоки	10
4.4.1. Введение	10
4.4.2. addCustomBlocks() и removeCustomBlock()	11
4.4.3. safeCustomBlocks()	11
4.4.4. safeTagChars()	11
4.4.5. buildSafedTag()	11
4.4.6. getCustomBlocks()	12
5. Приложение	13

1. Начало работы

1.1. Вводная часть

Класс `Jare_Typograph` предназначен для типографирования текста с учетом норм, правил и специфики русского языка. Для его работы необходим интерпретатор PHP версии 5.0 или выше, а так же текст в кодировке UTF-8.

Возможно два способа инициализации — быстрый и обычный. Если необходима гибкая настройка и адаптация `Jare_Typograph` под какую-либо систему, стоит начать с пункта 1.2.2, пропустив быструю инициализацию.

1.2. Инициализации

1.2.1. Быстрая

Данный способ инициализации класса `Jare_Typograph` позволяет быстро типографировать текст, при этом к нему будет применена вся стандартная цепочка типографирования:

```
require_once 'Jare/Typograph.php';
$text = Jare_Typograph::quickParse($text);
```

1.2.2. Обычная

Обычная инициализация позволяет полностью настроить цепочку типографирования: какие тофы и их параметры будут применены к тексту. При этом появляется возможность добавления своих собственных тофов в цепочку типографирования, а так же расширять или изменять существующий набор параметров у них.

```
// Инициализация веб-типографа
require_once 'Jare/Typograph.php';
$jareTypo = new Jare_Typograph($text);

// Типографирование текста
$text = $jareTypo->parse(array('etc', 'quote'));
```

1.2.2.1 Ключевые методы

1.2.2.1.1. `getBaseTofsNames()`

`getBaseTofsNames()` возвращает список названий тофов, которые идут в стандартной поставке.

1.2.2.1.2. `getTof()` и `setTof()`

`setTof($name, $object)` позволяет зарегистрировать тоф в цепочке типографирования, при этом объект должен быть наследником абстрактного класса `Jare_Typograph_Tof`.

```
/**
 * Тоф
 *
 * @author      Ivanov Ivan <somebody@domain.com>
 * @copyright   Copyright 2009 My Company
 */
class My_Tof extends Jare_Typograph_Tof
{
    // ...
}
```

```
// Инициализация веб-типографа
require_once 'Jare/Typograph.php';
$typograph = new Jare_Typograph('...');

// Инициализация тофа
$tof = new My_Tof();

// Регистрация тофа в цепочке типографирования
$typograph->setTof('my_tof', $tof);
```

getTof(\$name) позволяет получить объект тофа. Если тоф не был зарегистрирован с помощью **setTof()** и является стандартным, он будет зарегистрирован автоматически.

```
// Получение тофа
$tof = $typograph->getTof('my_tof');

if ($tof->isDisabledParsing()) {
    // ...
}
```

Подробнее о работе с тофами и их создание — второй раздел.

1.2.2.13. parse()

parse(\$tofs) осуществляет типографирование текста тофами. Перед типографированием все теги и содержимое безопасных блоков кодируется, что позволяет сохранить их целостность (например, предотвратить замену кавычек тега `` на коды «ёлочек»).

```
// Инициализация
require_once 'Jare/Typograph.php';
$typograph = new Jare_Typograph($text);

// Настройка тофов, их параметров
// ...

// Типографирование текста тофом
// с названием 'space'
$typograph->parse('space');

// Типографирование текста тофами
// 'etc' и 'space'
$typograph->parse(array('etc', 'space'));

// Трюк: типографирование текста всеми
// стандартными тофами
$typograph->parse($typograph->getBaseTofsNames());
```

2. Тофы

2.1. Обзор

Любой тоф в цепочке типографирования — это самостоятельная и независимая единица типографирования текста. Он может включать в себя набор параметров, правил типографирования, отобранных по критериям, характерным для конкретного тофа.

Идея, которая заложена в тофах — это объединение большого количества различных операций над одним объектом в группу.

2.2. Ключевые методы

2.2.1. `getBaseParam()` и `setBaseParam()`

`getBaseParam()` используется для получения параметра тофа с целью его изменения.

```
$oParam = $oJareTypograph->getTof('etc')->getBaseParam('some_param');
```

В результате выполнения этого примера будет получен экземпляр класса `Jare_Typograph_Param`, который является оболочкой для настройки запрошенного параметра.

`setBaseParam($name, Jare_Typograph_Param $param)` осуществляет расширение набора параметров тофа, а так же изменение его существующих параметров.

```
require_once 'Jare/Typograph/Param.php';  
$oParam = new Jare_Typograph_Param();  
$oParam->setOption('pattern', '/\040+/');  
$oParam->setOption('pattern', '');
```

```
$oJareTypograph->getTof('space')->setBaseParam('param_name', $oParam);
```

2.2.2. `disableParsing()` и `isDisabledParsing()`

`disableParsing($status)` позволяет отключить тоф — в процессе типографирования к тексту не будут применены параметры, которые содержатся в тофе.

```
$oJareTypograph->getTof('etc')->disableParsing(true);
```

`isDisabledParsing()` возвращает статус типографирования текста тофом.

2.2.3. `disableBaseParam()`

Иногда необходимо исключить обработку текста определенными параметрами тофа, при этом не отключая весь тоф.

`disableBaseParam()` принимает переменную типа `string` или `array` с именами параметров, которые будут проигнорированы в процессе типографирования:

```
$tof = $oJareTypograph->getTof('etc');  
  
// Отключение одного параметра  
$tof->disableBaseParam('param_1');  
// Отключение списка параметров  
$tof->disableBaseParam(array('param_2', 'param_3'));
```

2.3. Продвинутое использование

2.3.1. Создание собственных тофов

Одно из основных требований при создании тофа: он должен быть наследником абстрактного класса `Jare_Typograph_Tof`:

```
require_once 'Jare/Typograph/Tof.php';

class My_Tof extends Jare_Typograph_Tof
{
    /**
     * Базовые параметры тофа
     *
     * @var array
     */
    protected $_baseParam = array(
        'auto_times_x' => array(
            // Маска
            'pattern'      => '/(\d+) (\040*) (x|x) (\040*) (\d+)/u',
            // Замена
            'replacement' => '\1&times;\5'),
    );
}
```

Все параметры тофа обрабатываются в методе `parse()`. Если вы хотите внести изменения в процесс обработки, при этом они являются не существенными, вы можете перегрузить методы `_preParse()` и `_postParse()`, которые вызываются перед и после процесса обработки текста тофом соответственно:

```
require_once 'Jare/Typograph/Tof.php';

class My_Tof extends Jare_Typograph_Tof
{
    /**
     * Подготавливаем текст перед
     * процессом обработки
     *
     * @return void
     */
    protected function _preParse()
    {
        $this->_text = '<!--typo_start-->' . $this->_text;
    }

    /**
     * Сюда мы попадаем после типографирования
     * текста данным тофом
     *
     * @return void
     */
    protected function _postParse()
    {
        $this->_text .= '<!--typo_start-->';
    }
}
```

Параметры тофа могут ссылаться на его метод. Для этого у параметра должен быть установлен ключ `function_link`, в значение которого указано имя метода (при этом этот метод не может принимать никаких параметров):

```
require_once 'Jare/Typograph/Tof.php';

class My_Tof extends Jare_Typograph_Tof
{
```

```

/**
 * Базовые параметры тофа
 *
 * @var array
 */
protected $_baseParam = array(
    // Параметр
    'math_chars' => array(
        // Ссылка на метод класса
        'function_link' => '_buildMathChars'),
    );

/**
 * Расстановка простейших математических знаков
 *
 * @return void
 */
protected function _buildMathChars()
{
    $this->_text = str_replace('!=', '&ne;', $this->_text);
    $this->_text = str_replace('<=', '&le;', $this->_text);
    $this->_text = str_replace('>=', '&ge;', $this->_text);
    $this->_text = str_replace('~=', '&cong;', $this->_text);
    $this->_text = str_replace('+-', '&plusmn;', $this->_text);
}
}

```

3. Параметры тофов

3.1. Обзор

При типографировании текста основная часть операций приходится на регулярные выражения. Это связано с возможностью гибкого анализа текста, что крайне важно для применения правильной типографики.

В `Jare_Typograph` каждое регулярное выражение является параметром тофа (раннее регулярные выражения назывались правилами типографирования). Помимо этого, каждый параметр включает в себя ряд дополнительных опций, которые могут быть полезны при настройке.

Для работы с параметрами тофа используется специальный класс-оболочка `Jare_Typograph_Param`, речь о котором пойдет ниже.

3.2. Ключевые методы

3.2.1. `getOption()` и `setOption()`

Как было отмечено ранее, каждый параметр тофа включает в себя набор опций.

`setOption($name, $value)` позволяет задавать опции. В настоящее время тофы могут работать со следующими именами опций:

- `Jare_Typograph_Param::KEY_DISABLE_USER` — отключение типографирования текста данным параметром. Так же есть метод `disable()`, который выполняет ту же роль (см. раздел 3.2.3);
- `Jare_Typograph_Param::KEY_PARSE_PATTERN` — правила поиска (маска регулярного выражения);
- `Jare_Typograph_Param::KEY_PARSE_REPLACE` — шаблон замены найденного в процессе поиска;
- `Jare_Typograph_Param::KEY_FUNCTION_LINK` — имя метода тофа, не принимающего параметров, который будет вызван при типографировании текста тофом. Помимо этого, метод должен находиться в том же тофе, что и данный параметр. При использовании данной опции `Jare_Typograph_Param::KEY_PARSE_PATTERN` и `Jare_Typograph_Param::KEY_PARSE_REPLACE` сбрасываются, поскольку подразумевается, что весь процесс типографирования этим параметром вынесен в указанный метод. Использование `Jare_Typograph_Param::KEY_FUNCTION_LINK` актуально в тех случаях, когда параметр тофа имеет сложную структуру и не укладывается в одно регулярное выражение.

Далее показан простой пример, в котором задается правило поиска и шаблон замены: два и более пробелов в тексте будут заменены на один.

```
require_once 'Jare/Typograph/Param.php';
$params = new Jare_Typograph_Param();

// Задаем маску
$params->setOption(Jare_Typograph_Param::KEY_PARSE_PATTERN, '/(\040){2,}/');
// ... и замену
$params->setOption(Jare_Typograph_Param::KEY_PARSE_REPLACE, '/\1/');
```

`getOption($name)` служит для получения ранее заданного значения опции:

```
// Получаем значение по ключу - имя вызываемого метода
$params->getOption(Jare_Typograph_Param::KEY_FUNCTION_LINK);
```

3.2.2. reset()

reset() позволяет сбросить все сделанные значения к первоначальным. Использование метода возможно только при изменении существующего параметра тофа, но не создание нового.

```
// Получаем параметр для изменения
$param = $jareTypograph->getTof('etc')->getBaseParam('some_param');

// Задаем имя вызываемого метода
// Если изначально была указана макса и замена, то они будут удалены
$param->setOption(Jare_Typograph_Param::KEY_FUNCTION_LINK, '_parseLongData');

// Теперь все настройки имеют первоначальные значения
$param->reset();
```

3.2.3. disable()

disable(\$status) позволяет отключить типографирование текста данным параметром.

4. Инструментарий

4.1. Обзор

Как правило, текст, который необходимо типографировать, содержит в себе HTML-теги, различные специальные символы (например, скопированные из офисного пакета или другого приложения), а так же текст, который не должен быть типографирован. Для решения этих проблем был создан класс `Jare_Typograph_Tool`, которые содержит статические методы для организации текста перед и после типографирования.

4.2. `clearSpecialChars()`

`Jare_Typograph_Tool::clearSpecialChars($text, $mode)` осуществляет очистку текста от HTML кодов и специальных символов на основе таблицы. После его отработки все коды будут заменены на сущности, которые можно ввести с клавиатуры (например, код ` ` будет заменен на пробел, `&mdash`; — на дефис).

```
// Текст для очистки
$text = 'Этот &laquo;текст» будет очищен © Ivanov Co';

require_once 'Jare/Typograph/Tool.php';
// Удаляем коды...
$text = Jare_Typograph_Tool::clearSpecialChars($text);

// Выведет:
// Этот "текст" будет очищен (c) Ivanov Co
var_dump($text);
```

Данный метод вызывается автоматически при типографирование текста.

4.3. `removeHtmlTags()`

`Jare_Typograph_Tool::removeHtmlTags($text, $allowableTag = null)` удаляет HTML теги из текста. При этом все теги принудительного переноса (`
`) будут заменены на код переноса строки (`\n`), а закрывающий и отрывающий теги параграфа (`</p><p>`) — на два переноса.

При наличие тегов в тексте не всегда возможно корректное типографирование. Например, текст «Лес, `в` котором мы собирали ягоды» не будет типографирован правильно, потому что предлог «в» заключен в тег (между предлогом и словом должен быть неразрывный пробел).

```
require_once 'Jare/Typograph/Tool.php';

// Будут удалены все теги, кроме <a> и <span>
$text = Jare_Typograph_Tool::removeHtmlTags($text, array('a', 'span'));

// Будут удалены все теги, кроме <nobr>
$text = Jare_Typograph_Tool::removeHtmlTags($text, 'nobr');

// Будут удалены все теги без исключения
$text = Jare_Typograph_Tool::removeHtmlTags($text);
```

4.4. Безопасные блоки

4.4.1. Введение

Порой при типографирование текста встречаются участки, которые не должны быть типографированы. Как правило, они заключаются в специальные теги или ВВ-коды (например, `<pre>...</pre>` или `[pre]...[pre]`). Так как предусмотреть все возможные комбинации невозможно, в `Jare_Typograph_Tool` был создан механизм безопасных блоков, содержимое которых в процессе

типографирования кодируется алгоритмом BASE64, что приводит к невозможности типографирования этих участков кода.

4.4.2. addCustomBlocks() и removeCustomBlock()

Jare_Typograph_Tool::addCustomBlocks(\$open, \$close, \$quoted = false) позволяет добавлять безопасные блоки перед процессом типографирования. Если третий необязательный параметр имеет значение true, то специальные символы в начале и конце блока не будут экранированы (необходимо, если подставляется готовая маска поиска).

```
require_once 'Jare/Typograph/Tool.php';
// Внутри этого тега и ВВ-кода текст не будет типографирован
// Он будет кодирован...
Jare_Typograph_Tool::addCustomBlocks('<style>', '</style>');
Jare_Typograph_Tool::addCustomBlocks('[clean]', '[/clean]');

// В данном теге все специальные символы были экранированы в ручную
Jare_Typograph_Tool::addCustomBlocks('\<script>', '\</script>', true);

// Типографирование...
require_once 'Jare/Typograph.php';
$typograph = new Jare_Typograph('...');
```

Перед типографированием автоматически в список безопасных блоков добавляются теги <style>, <script> и <pre>.

Jare_Typograph_Tool::removeCustomBlock(\$blockId) позволяет удалить безопасный блок по его индексу.

4.4.3. safeCustomBlocks()

Jare_Typograph_Tool::safeCustomBlocks(\$text, \$safe) сохраняет текст внутри безопасных блоков. Он должен быть вызван перед методом **Jare_Typograph_Tool::safeTagChars()** до процесса типографирования и после него — после типографирования.

Он вызывается автоматически при типографирование текста.

4.4.4. safeTagChars()

Jare_Typograph_Tool::safeTagChars(\$text, \$safe) позволяет сохранять содержимое тегов — то, что находится внутри треугольных скобок. Данный метод вызывается автоматически перед, и после процесс типографирования.

```
$text = '<b>Текст</b> <nobr>для типографирования</nobr>';

// Сохраняем содержимое тегов
// Все, что находится между < и > будет кодировано
// - в данном примере: nobr, b
$text = Jare_Typograph_Tool::safeTagChars($text, true);

// ...

// Декодируем содержимое безопасных блоков
// Теги вносъ имеют первозданный вид
$text = Jare_Typograph_Tool::safeTagChars($text, false);
```

4.4.5. buildSafeTag()

Многие тофы и их параметры при типографирование текста вставляют HTML-теги.

Jare_Typograph_Tool::buildSafeTag(\$content, \$tag = 'span', \$attribute = array()) позволяет создавать защищенные теги, что позволит не нарушить их целостность при типографирование текста — это наиболее актуально при разработке собственных тофов с параметрами, имеющими сложную структуру (например, когда параметры ссылаются на методы тофа).

```
// Данный текст будет обрамлен тегом, при этом
// он (текст) не будет защищен - остальные тофы
// смогут типографировать его
$text = 'Какой-то текст';

// Тег без атрибутов
$tagNobr = Jare_Typograph_Tool::buildSafedTag($text, 'nobr');

// Тег с атрибутами 'id' и 'class'
$tagSpan = Jare_Typograph_Tool::buildSafedTag($text, 'span', array('class' =>
'my-css', 'id' => 'text-123'));
```

4.4.6. getCustomBlocks()

Jare_Typograph_Tool::getCustomBlocks() позволяет получить список всех добавленных безопасных блоков. Все специальные символы будут экранированы.

5. Приложение

Перечень стандартных тофов и их параметров.

Dash

mdash, mdash_2, mdash_3

три параметра `mdash` служат для расстановки длинного тире

years

расстановка короткого тире в интервалах дат. Для исключения ошибок, когда вместо интервала дат может быть записан математический пример, перед интервалом дат должно идти одно из слов: с, по, период, середины, начала, начало, конца, конец, половины, в, между.

iz_za_pod

расстановка дефиса между **из-за**, **из-под**

to_libo_nibud

расстановка дефиса перед **-то**, **-либо**, **-нибудь**

Ets

tm_replace

замена (tm) на код торговой марки, при этом лишние позади идущие пробелы будут удалены

r_sign_replace

замена (r) на зарегистрированный знак — будет заключен в теги `<small>` и `<sup>`

copy_replace

замена (c) на код авторского права, лишние спереди идущие пробелы будут удалены. Так же работает в случае, когда вместо английской буквы «с» указана русская

acute_accent

замена символа ` после гласных букв русского алфавита на ударение

auto_links

обрамление ссылок тегом гиперссылки `<a>` — возможно для протоколов `http`, `https`, `ftp`, а так же `mailto`

email

обрамление адресов электронной почты тегом гиперссылки `<a>`

hyphen_nowrap

заклочение слов с дефисом тегом ``, запрещающим разрыв таких слов в месте дефиса

simple_arrow

замена `->` и `<-` на коды стрелок

ip_address

объединение IP-адресов в неразрывные конструкции (только для IPv4). Так же учитывается разрядность — `567.128.128.0` не будет объединено

optical_alignment

оптическое выравнивание для знаков пунктуации (за исключением кавычек)

paragraphs

расстановка тегов параграфа `<p>` и принудительного переноса строк `
`

Numbers

auto_times_x

расстановка знака × между цифрами

numeric_sub

индексы

numeric_sup

степени

simple_fraction

расстановка простых степеней $-1/2$, $1/4$, $3/4$ (по умолчанию выключено)

math_chars

расстановка элементарных математических знаков — не равно, больше или равно, меньше или равно, плюс-минус...

Punctmark

auto_comma

расстановка запятой перед **а**, **но**

punctuation_marks_limit

удаление четырех и более подряд идущих повторяющихся знаков препинания. Например, «!!!!!» будет заменено на «!!!». Распространяется на восклицательный и вопросительный знаки, а так же точку.

punctuation_marks_base_limit

замена 2 и более подряд идущих запятых, точек с запятой, а так же двоеточия на один

hellip

расстановка многоточия

eng_apostrophe

расстановка апострофа в английских словах

fix_brackets

удаление пробелов внутри скобок

Quote

quotes_outside_a

вынос кавычек за гиперссылку

open_quote

расстановка открывающих кавычек

close_quote

расстановка закрывающих кавычек. Две и более подряд идущих кавычек будут заменены на одну.

optical_alignment

оптическое выравнивание открывающей кавычки

Space

nobr_abbreviation

расстановка неразрывного пробела перед сокращениями **dpi**, **lpi**

nobr_acronym

привязка сокращений **гл.**, **стр.**, **рис.** и **ил.** к числу

nobr_before_unit

привязка сокращений **м**, **мм**, **см**, **км**, **гм**, **km**, **dm**, **cm**, **mm** к числу

remove_space_before_punctuationmarks

удаление пробелов перед знаками препинания — запятая, точка и двоеточие

autospace_after_comma

расстановка пробела после запятой

autospace_after_pmarks

расстановка пробела после прочих знаков препинания

super_nbsp

расстановка неразрывных предлогов между предлогами

many_spaces_to_one

замена множества пробелов и табуляции на один пробел

clear_percent

удаление пробелов между числом и знаком процента

nbsp_before_open_quote

расстановка неразрывного предлога между предлогом и открывающей скобкой

nbsp_before_particle

привязка частиц **ли**, **бы**, **б**, **же**, **ж** к слову

ps_pps

объединение сокращений **P.S.** и **P.P.S.**