Gulp. Шаблонизаторы

06 DECEMBER 2017

Написание HTML-разметки документа - самая простая часть веб-разработки. Однако, даже к ней стоит подходить с умом. Любая HTML-структура содержит некий набор повторяющихся элементов. И их все можно объединить в один.

Версия без оптимизации (*Вид 1*):

```
<ul>
  <li>
    <h1 class="name">Егор</h1>
    <h2>Васильев</h2>
    <span class="date">10.09.1990</span>
  </li>
  <li>
    <h1 class="name">Валентин</h1>
    <h2>Фролов</h2>
    <span class="date">11.03.1992</span>
  </li>
  <li>
    <h1 class="name">Мария</h1>
    <h2>Мышкина</h2>
    <span class="date">20.11.1989</span>
  </li>
</ul>
```

Версия с оптимизацией (*Вид 2*):

```
<ul>
  {{#each items}}
  <li>
    <h1 class="name">{{name}}</h1>
    <h2>{{fullname}}</h2>
    <span class="date">{{date}}</span>
  </li>
```

```
  {/each}}
</ul>
```

Мы только что создали шаблон для каждого пользователя. После того, как мы подгрузим в него нужные данные, он преобразуется к *Виду 1*. Подобные структуры вместе с переданными данными называются **шаблонизаторами**.

Шаблонизатор PUG

Шаблонизаторы бывают разных видов. Наиболее известными являются Pug, Handlebar и Jade. Отличаются друг от друга они, прежде всего, синтаксисом. Мы рассмотрим первый из них.

1.Подключение.
Подключать шаблонизатор мы будем через *gulp*. Выполняем команду:

```
npm i gulp-pug --save-dev
```

2.Меняем Gulp-файл.

```
// Добавляем новую переменную
    ...
    pug = require('gulp-pug');
    ...

// Создаем НОВЫЙ таск

gulp.task('templates', function buildHTML() {
    return gulp.src('./templates/pages/*.pug') // возьми все файлы по
        .pipe(pug({
            pretty: true // в объекте указываются дополнительные настр
        }).on('error', function(error) {
            console.log(error); // если нашел ошибку при компиляции, п
        }))
        .pipe(gulp.dest('build')); // положи результат в эту папку
});
```

3.Тренировка.
Давайте поэксперементируем! Создадим главную страницу в папке */templates/pages*.

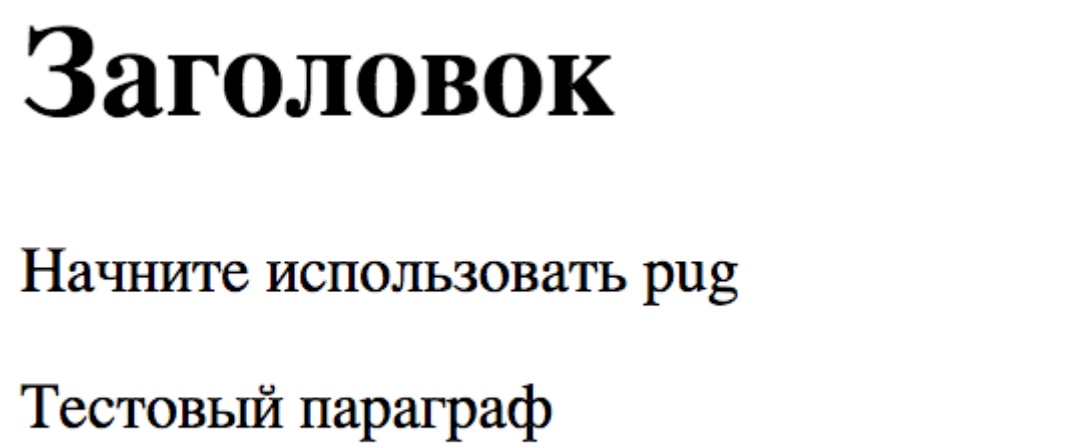
```
// index.pug
doctype html
html(lang="en")
    head
        title='Тестовая страница'
    body
        h1 Заголовок
        #container.col
            if youAreUsingPug
                p Вы используете pug
            else
                p Начните использовать pug
            p.
                Тестовый параграф
```

Синтаксис у *pug*, конечно, специфический. Но мы рассмотрим его позднее.

Запускаем нашу команду:

```
gulp templates
```

И в папке *build* наблюдаем index.html - результат нашей работы:



Кстати, его код выглядит как чистый *HTML*:

```
// build/index.html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Тестовая страница</title>
  </head>
  <body>
    <h1>Заголовок</h1>
    <div class="col" id="container">
      <p>Начните использовать pug</p>
      <p>Тестовый параграф</p>
    </div>
  </body>
</html>
```

Шаблонизатор преобразуем весь тот сложный код, что мы написали выше *(index.pug)* к стандартному понятному виду.

4.Замена HTML на PUG

Отлично, таск для *gulp* мы сформировали. Теперь нам нужно сделать так, чтобы он работал для всего проекта, заменив собою таск с HTML.

Для этой цели меняем *gulpfile.js.*

```
// Удаляем таск html

gulp.
gulp.task('watch', function() {
    ...
    gulp.watch('./templates/**/*.pug', gulp.series('templates'));
    ...
});
});

// Во всех тасках, где используется html, меняем его на templates
gulp.task('default', ['clean'], function() {
    ...
    gulp.run('templates');
    ...
});
```

Запускам командой `gulp`. УРА!!! Теперь шаблонизатор подключем ко всему проекту.

Структура PUG

Теперь немного о структуре самого *Pug*. Обычно сайт состоит из множества страниц, в которых есть одинаковые части (логотип, футер и др.). Эти одинаковые части можно вынести в общей шаблон.

Для этого внутри *templates* создадим папку *layout* с файлом *main.pug*, в котором укажем:

```
// main.pug
doctype html
html(lang="en")
    head
        title='Тестовая страница'
    body
        block main // в этот блок будем загружать данные по каждой стр
```

А сами, конкретные страницы, по-прежнему оставим в *pages*. Например, */pages/test.pug*

```
// pages/test.pug
extends ../layout/main // сначала грузим все общее, что есть в layout

block main // затем добавляем содержимое в блок main
    #container.col
        if youAreUsingPug
            p Вы используете pug
        else
            p Начните использовать pug
        p.
            Тестовый параграф
```

Перейдем по адресу http://localhost:3000/test.html. Результат тот же самый:

```
// build/test.html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Тестовая страница</title>
  </head>
  <body>
```

```
    <h1>Заголовок</h1>
    <div class="col" id="container">
      <p>Начните использовать pug</p>
      <p>Тестовый параграф</p>
    </div>
  </body>
</html>
```

Согласно команде *extends ../layout/main* мы расширяли содержимое дочернего файла *pages/test.pug*. Но можно также осуществлять и *include*, т.е., наоборот, писать в родительском файле, что мы грузим дочерний. Обычно для этого случая все дочерние элементы выносят в папку *partials*.

Наша структура:

```
gulp-project
-- src
   -- templates
      -- layout
      -- pages
      -- partials
```

Давайте в рамках partials вынесем общий *header*.

```
// partials/header.pug
header.header
    nav.header-nav
        a.header-nav__logo(href="/")
            .sprite__logo
```

И сделаем *include*.

```
doctype html
html(lang="en")
    head
        title='Тестовая страница'
    body
        block header // создали еще один блок header
            include ../partials/header // и подключили в него содержим
        block main
```

Результат:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Тестовая страница</title>
  </head>
  <body>
    <header class="header">
      <nav class="header-nav"><a class="header-nav__logo" href="/">
          <div class="sprite__logo"></div></a></nav>
    </header>
    <div class="col" id="container">
      <p>Начните использовать pug</p>
      <p>Тестовый параграф</p>
    </div>
  </body>
</html>
```

Синтаксис PUG

При работе с *PUG* следует обращать внимание на синтаксис как никогда. В особенности, на отступы.

Вся суть разметки *PUG* сводится к проставлению правильных отступов.

```
doctype html
html(lang="en") // без отступа - элемент самого верхнего уровня
    head // одинарный отступ - элемент внутри html
        title='Тестовая страница' - двойной отступ - элемент внутри пр
    body // одинарный отступ - элемент на одном уровне с body
        block header // создали еще один блок header // двойной отступ



            include ../partials/header // тройной отступ, элемент внут
        и т.д. и т.п.
```

Названия классов и атрибуты записывается через подобный синтаксис:

```
// partials/header.pug
    nav.header-nav // присвой элементу nav класс header-nav
    a.header-nav__logo(href="/") // создай ссылку с классом header-nav
```

Важной особенностью *PUG* является возможность работы с переменными:

```
    - var pageTitle = 'Home' // переменная pageTitle
    - var pageDescription = 'Home descr' // переменная pageDescriptior


// подобные js-конструкции пишем через символ "-"
...
html(lang="ru")
    head
        title #{pageTitle} // отображаем переменную  pageTitle
        meta(name="description" content=pageDescription) // отображаем
```

И с циклами:

```
- for (var x = 0; x < 3; x++)
  li item
```

преобразуется к виду:

```
<li>item</li>
<li>item</li>
<li>item</li>
```

А если передадим содержимое

```
-
  var list = ["Uno", "Dos", "Tres",
          "Cuatro", "Cinco", "Seis"]
each item in list
  li= item
```

то к этому:

```
<li>Uno</li>
<li>Dos</li>
<li>Tres</li>
<li>Cuatro</li>
<li>Cinco</li>
<li>Seis</li>
```

Подробнее со всей документаций *PUG* можно ознакомиться по ссылке.

@Afelua

Read more posts by this author.

Moscow

Share this post