Web Standards Days

DOM-шаблонизаторы – не только «быстро»

Роман Дворнов

Ostrovok.ru

Июнь 2014

О себе

- Работаю в ostrovok.ru
- Автор basis.js

Сегодня мы создаем

динамические интерфейсы

Динамические интерфейсы

=

работа с DOM

Построение интерфейса

=

создание DOM-фрагментов

и их композиция

Что можно использовать для создания DOM-фрагментов?

Что можно использовать для создания DOM-фрагментов?

- Руки ;)

Что можно использовать для создания DOM-фрагментов?

- Руки ;)
- String-шаблонизаторы

Что можно использовать для создания DOM-фрагментов?

- Руки ;)
- String-шаблонизаторы
- DOM-шаблонизаторы

String vs. DOM

String-шаблонизаторы

Описание шаблона =

набор строк + инструкции

String-шаблонизаторы

Описание (handlebars)

```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```

Интерпретация

```
… + title +
… + body +
…
```

Не известно какая в итоге получится DOM-структура и как она будет меняться

Зато достаточно

универсальный подход,

а HTML частный случай

Строковые шаблонизаторы производят функции, которые производят строки

После этого

innerHTML + пост-процессинг

Пост-процессинг

- Получение ссылок на элементы
- Навешивание обработчиков событий
- Модификация DOM
  (использование jQuery-like плагинов,
  вставка DOM фрагментов etc)
- и т.д.

Пост-процессинг

самая медленная фаза

Процесс

Создание

- геререация HTML
- innerHTML
- пост-процессинг

Обновление

- генерация HTML
- innerHTML
- пост-процессинг

Процесс

Создание

- геререация HTML
- innerHTML
- пост-процессинг

=

Обновление

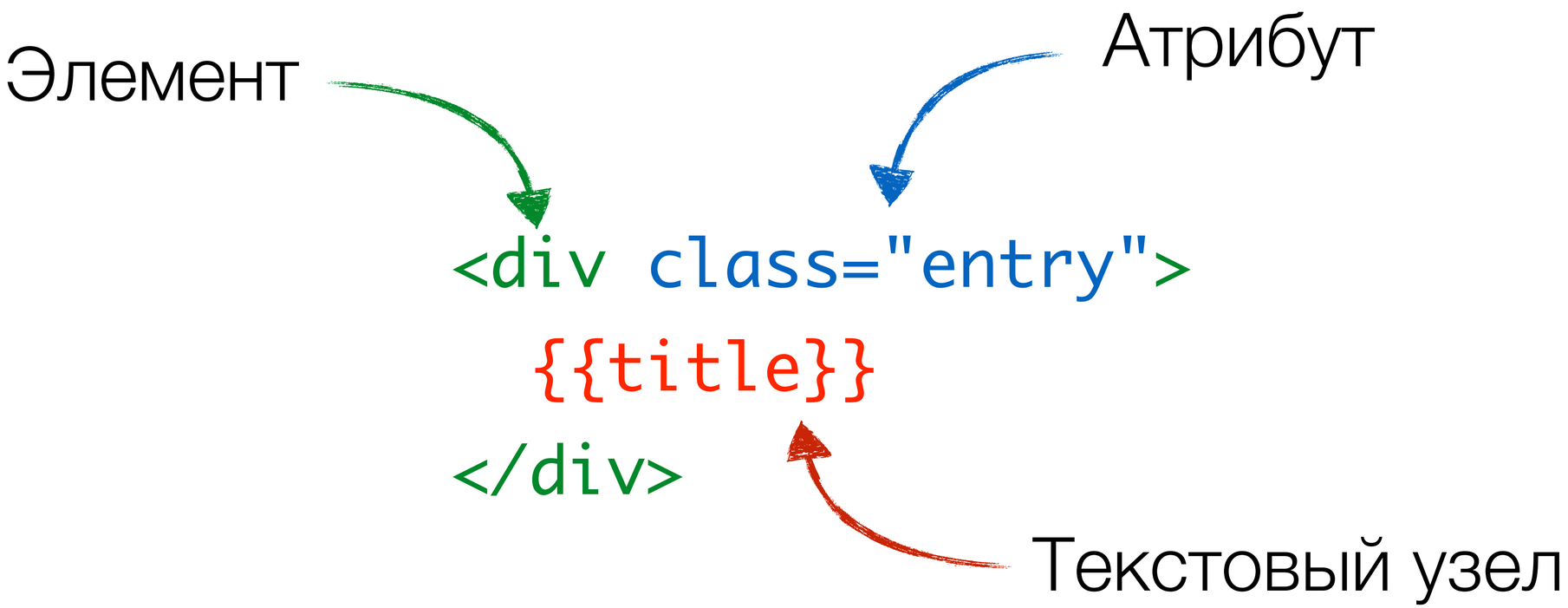
- генерация HTML
- innerHTML
- пост-процессинг

DOM-шаблонизаторы

Описание шаблона =

DOM узлы + инструкции

DOM шаблонизаторы



Известно какая будет DOM-структура и как будет меняться

… еще до построения

нативного DOM

Ориентированы на работу с DOM*

* Но не обязательно производят только DOM

DOM-шаблонизаторы производят и обслуживают DOM-фрагменты

Знания о DOM структуре позволяют использовать оптимизации

Например

- cloneNode(true) – быстрое создание DOM-фрагмента
- обработка событий через один глобальный capture-обработчик на документе для каждого уникального события

Процесс

Создание

- построение DOM-фрагмента или cloneNode(true)
- DOM-операции

Обновление

- DOM-операции

Работают быстрее

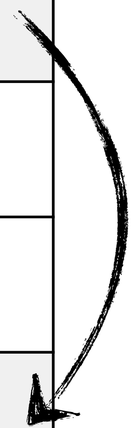
Могут проигрывать

при генерации больших фрагментов неизменяемой верстки

Но выигрывают на генерации повторяющихся фрагментов

TodoMVC

	100 items	1000 items
AngularJS	125 ms	1491 ms
Backbone	**53 ms**	**510 ms**
Knockout	39 ms	489 ms
jQuery	20 ms	184 ms
Backbone + basis.js templates	**18 ms**	**202 ms**
basis.js	8 ms	95 ms



Всегда выигрывают
на обновлении

Подробнее в докладе

Как построить DOM

tinyurl.com/build-dom

Уже используется в …

- Ractive
- Basis.js
- Ember: HTMLBars (в разработке)
- Meteor
- React*

* нет шаблонизатора, но внутри DOM-подход

Не холивара ради…

String

Универсальное решение для сайтов и приложений, подходит для сервера и клиента

DOM

Оптимально для больших динамических одностраничных приложений

String-шаблонизаторы

DOM-шаблонизаторы

Развивая идеи DOM-шаблонизации…

На примере basis.js

Абстрагирование от верстки

В коде

- Абстрагируемся от верстки
- Нет селекторов
- Нет указания CSS классов, имен тегов
- Нет HTML
- Нет стилей

Не знаем о разметке

и как используются значения

Работаем с интерфейсом

Описание

```
<div{el} class="example">
  {text}
</div>
```

Интерфейс к экземпляру

```
{
  el: <div>,
  text: #text,
  set: function(name, value){
    // black magic box
  }
}
```

Отдаем значения,
но что там происходит не знаем

Не знаем какие классы

в разметке, не используем

селекторы в JavaScript

Жизнь без селекторов

Описание

```
<div{el} class="example">
  <h1>{text}</h1>
  <ul{content}/>
</div>
```

Получение ссылок

(пути генерирует шаблонизатор)

```
el = fragment.firstChild
text = el.firstChild.firstChild
content = el.lastChild
```

Подключение стилей

<b:style>

Подключение стилей

block.tmpl

```
<b:style src="block.css"/>

<div class="block block_{hidden}">
  {caption}
</div>
```

block.css

```
.block
{
  ...
}
.block_hidden
{
  ...
}
```

Включение других шаблонов

<b:include>

Включение шаблонов

example.tmpl

```
<b:style src="./example.css"/>
<div class="wrapper">
  <b:include src="./button.tmpl">
    <b:after ref="caption"> {count}</b:after>
  </b:include>
</div>
```

button.tmpl

```
<b:style src="./button.css"/>
<button class="button">
  {caption}
</button>
```

Результат

```
<b:style src="./button.css"/>
<b:style src="./example.css"/>
<div class="wrapper">
  <button class="button">
    {caption} {count}
  </button>
</div>
```

Изоляция стилей

<b:isolate>

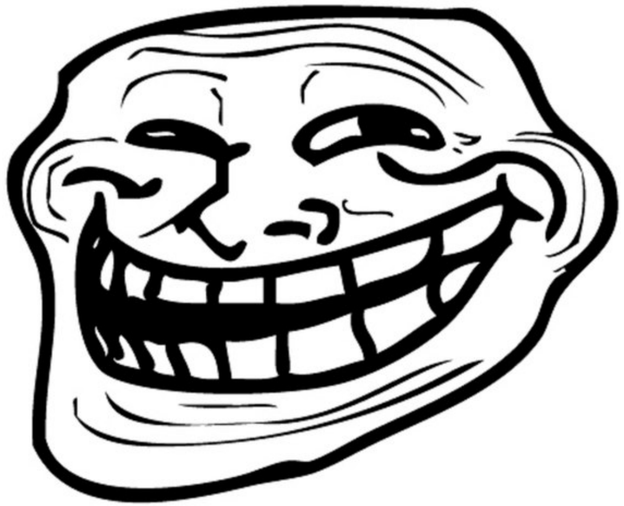
До

```
<b:style src="option.css"/>

<div class="xo-bookings-change-status-popup-option
xo-bookings-change-status-popup-option_{disabled}
xo-bookings-change-status-popup-option_{hidden}">
  <span class="xo-bookings-change-status-popup-
option__caption xo-bookings-change-status-popup-
option__caption_{selected}">
    {title}
  </span>
</div>
```

После

```
<b:style src="option.css"/>
<b:isolate/>

<div class="option {disabled} {hidden}">
  <span class="caption caption_{selected}">
    {title}
  </span>
</div>
```

В топку BEM?



В топку префиксы!

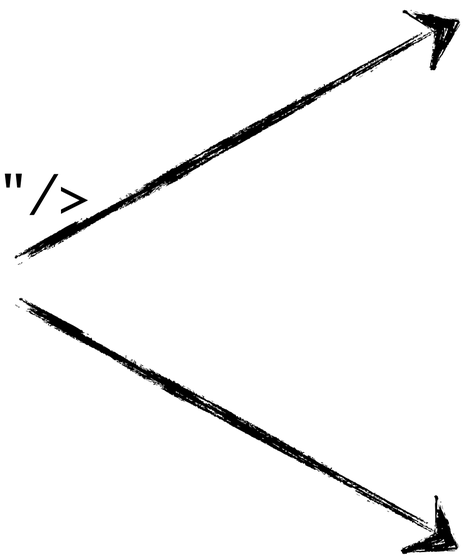
Но фишка не только в префиксах…

Переопределение стилей вместо добавления модификаторов

Безопасное дополнение стилей

button.tmpl

```
<b:style src="./button.css"/>
<button class="button">
  click me
</div>
```



ok-button.tmpl

```
<b:isolate/>
<b:style>
  .button { color: green; }
</b:style>
<b:include src="./button.tmpl"/>
```

cancel-button.tmpl

```
<b:isolate/>
<b:style>
  .button { color: red; }
</b:style>
<b:include src="./button.tmpl"/>
```

Изоляция включений

example.tmpl

```
<b:isolate/>
<b:style>
  .abc-foo { color: red; }
  .xyz-foo { color: blue; }
</b:style>
<div class="panel">
  <b:include src="./foo.tmpl" isolate="abc-"/>
  <b:include src="./foo.tmpl" isolate="xyz-"/>
</div>
```

foo.tmpl

```
<b:style src="./foo.css"/>
<div class="foo">
  example
</div>
```

Изоляция включений: результат

```
<b:style src="./foo.css?prefix=hs83shyf_abc-"/>
<b:style src="./foo.css?prefix=hs83shyf_xyz-"/>
<b:style>
  .hs83shyf_abc-foo { color: red; }
  .hs83shyf_xyz-foo { color: blue; }
</b:style>
<div class="hs83shyf_panel">
  <div class="hs83shyf_abc-foo">
    example
  </div>
  <div class="hs83shyf_xyz-foo">
    example
  </div>
</div>
```

Пространства имен

(Побочный продукт методики)

```
<b:style src="./bootstrap.css" ns="bt"/>
<b:style src="./glyphs.css" ns="glyph"/>
<b:style src="./style.css"/>

<div class="glyph:active bt:active active">
  ...
</div>
```

Гарантия уникальности классов

=

более сильные оптимизации

Например

example.tmpl

```
<b:isolate/>
<b:style src="./style.css"/>
<div class="foo bar">
  ..
</div>
```

style.css

```
.foo {
  color: red;
  width: 100px;
}
.bar {
  color: green;
}
```

Оптимизация: слияние стилей

example.tmpl

```
<b:isolate/>
<b:style src="./style.css"/>
<div class="baz">
  ..
</div>
```

style.css

```
.baz {
  /* color: red; */
  width: 100px;
  color: green;
}
```

И мы продолжаем

экспериментировать ;)

«Эээ… но у нас же есть Web Components…»

Кто-то из зала

Все таки их пока нет* ;)

* в процессе разработки и имплементации

Web Components – деклативная обертка для инкапсулированных ресурсов (DOM, CSS, JS)

В моем понимании

В простом случае:

1 DOM фрагмент

+

DOM-манипуляции*

* мог бы делать шаблонизатор

В комплексном случае:

Много фрагментов

+

возвращаемся к проблеме шаблонизации

Web Components

Хорошо для виджетов

(проигрыватели, карты и т.д)

Web Components

Не решает проблем самих компонент и приложений

Не совсем про шаблонизатор,

но мы храним

шаблоны в отдельных файлах

И остальные тоже скоро будут так делать ;)

Абстрагирование

+

внешние файлы

=

Live update

Live update – обновление DOM-фрагментов без перезагрузки страницы

Замена DOM-фрагмента

Старый DOM

```
<div class="sidebar">
  ...
  <ul class="list">
    <li>item 1</li>
    <li>item 2</li>
  </ul>
  ...
</div>
```

Новый DOM

```
<div class="list-wrapper">
  <h2>Header</h2>
  <ul class="list">

  </ul>
</div>
```

Замена DOM-фрагмента

Старый DOM

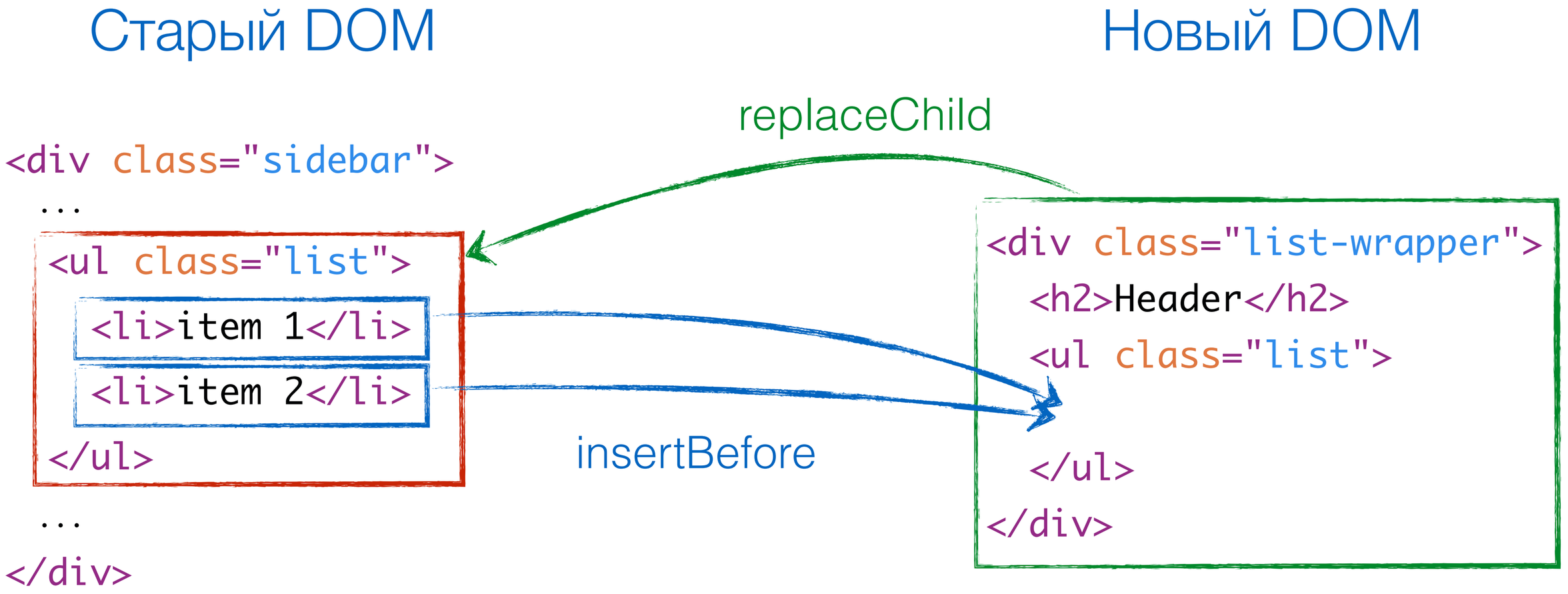
Новый DOM

```
<div class="sidebar">
  ...
  <ul class="list">
    <li>item 1</li>
    <li>item 2</li>
  </ul>
  ...
</div>
```

insertBefore

```
<div class="list-wrapper">
  <h2>Header</h2>
  <ul class="list">

  </ul>
</div>
```

Замена DOM-фрагмента



Live update

экономит время и

разгоняет разработку

Live update

+

Логика

=

Адаптивные View

Абстрагирование

+

Наборы шаблонов

=

Темы

Результат

```
basis.template.theme('base').define({
  foo: resource('path/to/file.tmpl'),
  ...
});
basis.template.theme('myTheme').define({
  foo: resource('path/to/file.tmpl'),
  ...
});

var view = new basis.ui.Node({
  template: basis.template.get('foo'),
  ...
});
```

Набор шаблонов базовой темы

Еще одна тема

Использование шаблона по имени

Тема = HTML + CSS

Тоже

без перезагрузки страницы

Но это побочный эффект ;)

Экземпляры шаблонов хранят мета-информацию*

* только в dev-режиме

Это дает возможность определять к какому шаблону относится DOM-узел

Heat map

показывает **где** и

как часто меняется DOM

Demo

github.com/lahmatiy/template-demos

Вне шаблонизатора

Анализ и выявление проблем

- какие классы используются в разметке, но их нет в стилях
- какие селекторы никогда не сработают
- конфликты стилей
- и т.д.

Оптимальная сборка

- Минимизация классов
- Удаление разметки и стилей, которые не используются
- и т.д.

И многое другое…

И все благодаря DOM

В строковых шаблонизаторах
многое из этого невозможно
или крайне сложно

DOM это круто!

Главное начать

думать в терминах DOM

Попробуйте ;)

Вопросы?

Роман Дворнов
@rdvornov
rdvornov@gmail.com

basis.js
basisjs.com
github.com/basisjs