Создание графических компонентов и объекты HTML5

Оглавление

[Синтаксис языка 0](#_Toc444164282)

[Внешние скрипты и порядок их исполнения 1](#_Toc444164283)

[Структура кода 2](#_Toc444164284)

[Комментарии 2](#_Toc444164285)

[Переменные 3](#_Toc444164286)

[Константы 3](#_Toc444164287)

[Типы данных 4](#_Toc444164288)

[Оператор typeof 5](#_Toc444164289)

[Современный стандарт, "use strict" 6](#_Toc444164290)

[Домашнее задание 6](#_Toc444164291)

[Используемая литература 7](#_Toc444164292)

# Графические компоненты

Первый и главный шаг в наведении порядка – это оформить код в объекты, каждый из которых будет решать свою задачу.

Здесь мы сосредоточимся на графических компонентах, которые также называют «виджетами».

В браузерах есть встроенные виджеты, например <select>, <input> и другие элементы, о которых мы даже и не думаем, «как они работают». Они «просто работают»: показывают значение, вызывают события…

Наша задача – сделать то же самое на уровне выше. Мы будем создавать объекты, которые генерируют меню, диалог или другие компоненты интерфейса, и дают возможность удобно работать с ними.

# Виджет menu

Мы начнём работу с виджета, который предусматривает уже готовую разметку.

То есть, в нужном месте HTML находится DOM-структура для меню – заголовок и список опций:

<div class="menu" id="sweets-menu">

<span class="title">СУБД</span>

<ul>

<li>MongoDB</li>

<li>MySQL</li>

<li>...</li>

</ul>

</div>

Далее она может дополняться, изменяться, но в начале – она такая.

Обратим внимание на важные соглашения виджета:

Вся разметка заключена в корневой элемент <div class="menu" id="sweets-menu">.

Это очень удобно: вынул этот элемент из DOM – нет меню, вставил в другое место – переместил меню. Кроме того, можно удобно искать подэлементы.

Внутри корневого элемента – только классы, не id.

Документ вполне может содержать много различных меню. Они не должны конфликтовать между собой, поэтому для разметки везде используются классы.

Исключение – корневой элемент. В данном случае мы предполагаем, что данное конкретное «меню СУБД» в документе только одно, поэтому даём ему id.

**Класс виджета**

Для работы с разметкой будем создавать объект new Menu и передавать ему корневой элемент. В конструкторе он поставит необходимые обработчики:

function Menu(options) {

var elem = options.elem;

elem.onmousedown = function() {

return false;

}

elem.onclick = function(event) {

if (event.target.closest('.title')) {

elem.classList.toggle('open');

}

};

}

// использование

var menu = new Menu({

elem: document.getElementById('sweets-menu')

});

.menu ul {

display: none;

margin: 0;

}

.menu .title {

font-weight: bold;

cursor: pointer;

}

.menu .title:before {

content: '▶';

padding-right: 6px;

color: green;

}

.menu.open ul {

display: block;

}

.menu.open .title:before {

content: '▼';

}

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<link rel="stylesheet" href="style.css">

<script src="https://cdn.polyfill.io/v1/polyfill.js?features=Element.prototype.closest"></script>

<script src="menu.js"></script>

</head>

<body>

<div id="menu" class="menu">

<span class="title">СУБД</span>

<ul>

<li>MongoDB</li>

<li>MySQL </li>

<li>PostgreSQL</li>

<li>RethinkDB</li>

<li>Redis</li>

</ul>

</div>

<script>

var menu = new Menu({

elem: document.getElementById('menu')

});

</script>

</body>

</html>

Это, конечно, только первый шаг, но уже здесь видны некоторые важные соглашения в коде.

У конструктора только один аргумент – объект options.

Это удобно, так как у графических компонентов обычно много настроек, большинство из которых имеют разумные значения «по умолчанию». Если передавать аргументы через запятую – их будет слишком много.

Обработчики назначаются через делегирование.

Вместо того, чтобы найти элемент и поставить обработчик на него:

var titleElem = elem.querySelector('.title');

titleElem.onclick = function() {

elem.classList.toggle('open');

}

…Мы ставим обработчик на корневой elem и используем делегирование:

elem.onclick = function(event) {

if (event.target.closest('.title')) {

elem.classList.toggle('open');

}

};

Это ускоряет инициализацию, так как не надо искать элементы, и даёт возможность в любой момент менять DOM внутри, в том числе через innerHTML, без необходимости переставлять обработчик.

В этот код лучше добавить дополнительную проверку на то, что найденный .title находится внутри elem:

elem.onclick = function(event) {

var closestTitle = event.target.closest('.title');

if (closestTitle && elem.contains(closestTitle)) {

elem.classList.toggle('open');

}

};

**Публичные методы**

Уважающий себя компонент обычно имеет публичные методы, которые позволяют управлять им снаружи.

Рассмотрим повнимательнее этот фрагмент:

if (event.target.closest('.title')) {

elem.classList.toggle('open');

}

Здесь в обработчике события сразу код работы с элементом. Пока одна строка – всё понятно, но если их будет много, то при чтении понадобится долго и упорно вникать: «А что же, всё-таки, такое делается при клике?»

Для улучшения читаемости выделим обработчик в отдельную функцию toggle, которая к тому же станет полезным публичным методом:

function Menu(options) {

var elem = options.elem;

elem.onclick = function(event) {

if (event.target.closest('.title')) {

toggle();

}

};

function toggle() {

elem.classList.toggle('open');

}

this.toggle = toggle;

}

Теперь метод toggle можно использовать и снаружи:

var menu = new Menu(...);

menu.toggle();

**Генерация DOM-элемента**

До этого момента меню «оживляло» уже существующий HTML.

Но далеко не всегда в HTML уже есть готовая разметка. В сложных интерфейсах намного чаще её нет, а есть данные, на основе которых компонент генерирует разметку.

В случае меню, данные – это набор пунктов меню, которые передаются конструктору.

Для генерации DOM добавим меню три метода:

render() – генерирует корневой DOM-элемент и заголовок меню.

renderItems() – генерирует DOM для списка опций ul/li.

getElem() – возвращает DOM-элемент меню, при необходимости запуская генерацию, публичный метод.

Функция генерации корневого элемента с заголовком render отделена от генерации списка renderItems. Почему – будет видно чуть далее.

Новый способ использования меню:

// создать объект меню с данным заголовком и опциями

var menu = new Menu({

title: "СУБД",

items: [

"MongoDB",

"MySQL",

"PostgreSQL",

"RethinkDB",

"Redis"

]

});

// получить сгенерированный DOM-элемент меню

var elem = menu.getElem();

// вставить меню в нужное место страницы

document.body.appendChild(elem);

Код Menu с новыми методами:

function Menu(options) {

var elem;

function getElem() {

if (!elem) render();

return elem;

}

function render() {

elem = document.createElement('div');

elem.className = "menu";

var titleElem = document.createElement('span');

elem.appendChild(titleElem);

titleElem.className = "title";

titleElem.textContent = options.title;

elem.onmousedown = function() {

return false;

};

elem.onclick = function(event) {

if (event.target.closest('.title')) {

toggle();

}

}

}

function renderItems() {

var items = options.items || [];

var list = document.createElement('ul');

items.forEach(function(item) {

var li = document.createElement('li');

li.textContent = item;

list.appendChild(li);

});

elem.appendChild(list);

}

function open() {

if (!elem.querySelector('ul')) {

renderItems();

}

elem.classList.add('open');

};

function close() {

elem.classList.remove('open');

};

function toggle() {

if (elem.classList.contains('open')) close();

else open();

};

this.getElem = getElem;

this.toggle = toggle;

this.close = close;

this.open = open;

}

Отметим некоторые особенности этого кода.

Обработчики отделяются от реальных действий.

В обработчике onclick мы «ловим» событие и выясняем, что именно произошло. Возможно, нужно проверить event.target, координаты, клавиши-модификаторы, и т.п. Это всё можно делать здесь же.

Выяснив, что нужно сделать, обработчик onclick не делает это сам, а вызывает для этого соответствующий метод. Этот метод уже не знает ничего о событии, он просто делает что-то с виджетом. Его можно вызвать и отдельно, не из обработчика.

Здесь есть ряд важных плюсов:

Обработчик onclick не «распухает» чрезмерно.

Код гораздо лучше читается.

Метод можно повторно использовать, в том числе и сделать публичным, как в коде выше.

Генерация DOM, по возможности, должна быть «ленивой».

Мы стараемся откладывать работу до момента, когда она реально нужна. Например, когда new Menu создаётся, то переменная elem лишь объявляется. DOM-дерево будет сгенерировано только при вызове getElem() функцией render().

Более того! Пока меню закрыто – достаточно заголовка. Кроме того, возможно, посетитель вообще никогда не раскроет это меню, так зачем генерировать список раньше времени? А при первом открытиии open() вызовет функцию renderItems(), которая специально для этого выделена отдельно от render().

Фаза инициализации очень чувствительна к производительности, так как обычно в сложном интерфейсе создаётся много всего.

Если изначально подходить к оптимизации на этой фазе «спустя рукава», то потом поправить долгий старт может быть сложно. Тем более, что инициализация – это фундамент, начало работы виджета, её оптимизация в будущем может потребовать сильных изменений кода.

Конечно, здесь, как и везде в оптимизации – без фанатизма. Бывают ситуации, когда гораздо удобнее что-то сделать сразу. Если это один элемент, то оптимизация здесь ни к чему. А если большой фрагмент DOM, который, как в случае с меню, прямо сейчас не нужен – то лучше отложить.

# Canvas

Если верить англо-русскому словарю, то можно узнать что canvas переводится как холст, а если верить википедии, то можно узнать что тег canvas, это элемент HTML 5, который предназначен для создания растрового изображения при помощи JavaScript.

Рассмотрим рисование на canvas на конкретном примере:

<!DOCTYPE html>

<html>

<head>

<title>Canvas</title>

<script type="text/javascript">

window.onload = function()

{

var elem = document.getElementById('canvas');

var context = elem.getContext('2d');

if (!context) {

alert('Скачай нормальный браузер');

return;

}

// Цвет обводки

context.strokeStyle = '#f00';

// Цвет заливки

context.fillStyle = '#ff0';

// Толщина линий

context.lineWidth = 4;

// Рисуем голову. Рисование всегда начинается с beginPath

context.beginPath();

// Размечаем дугу с центром в точке (150, 150), радиусом 100px, начальным углом 0, конечным 360 градусов, рисование производится по часовой стрелке. Иными словами получаем окружность

context.arc(150, 150, 100, 0, 2\*Math.PI, true);

// Обводим

context.stroke();

// Завершаем рисование

context.closePath();

// Начинаем рисование

context.beginPath();

// Рисуем окружность (глаз)

context.arc(110, 110, 15, 0, 2\*Math.PI, true);

// Закрашиваем

context.fill();

// Заканчиваем рисование

context.closePath();

// Рисуем второй глаз

context.beginPath();

context.arc(190, 110, 15, 0, 2\*Math.PI, true);

context.fill();

context.closePath();

// Рисуем нос

context.beginPath();

// Функция устанавливает точку из которой будет осуществляться рисование

context.moveTo(150, 120);

// Проводит линию из установленной точки, в указанную и так же устанавливает точку из которой будет осуществляться рисование

context.lineTo(135, 180);

context.lineTo(165, 180);

context.stroke();

context.closePath();

// Рот

context.beginPath();

// Половинка окружности

context.arc(150, 150, 70, Math.PI, 2\*Math.PI, true);

context.stroke();

context.closePath();

// Волосы – другим цветом

context.strokeStyle = '#000';

context.beginPath();

var x = 0;

var y = 0;

for(var i = 0; i < 5; i++)

{

context.moveTo(80 + x, 85 + y);

// Кривая безье

context.bezierCurveTo(60 + x, 60 + y, 120 + x, 40 + y, 100 + x, 0 + y);

context.stroke();

x += 15;

y -= 5;

}

for(var i = 0; i < 5; i++)

{

context.moveTo(80 + x, 85 + y);

context.bezierCurveTo(0 + x, 60 + y, 220 + x, 40 + y, 100 + x, 0 + y);

context.stroke();

x += 15;

y += 5;

}

context.closePath();/\* \*/

};

</script>

</head>

<body>

<canvas id="canvas" width="300" height="300" style = "border: 1px solid black">

</canvas>

</body>

</html>

# Домашнее задание

* Создайте компонент «Часы» (Clock).

Интерфейс:

var clock = new Clock({

elem: элемент

});

clock.start(); // старт

clock.stop(); // стоп

Остальные методы, если нужны, должны быть приватными.

При нажатии на alert часы должны приостанавливаться, а затем продолжать идти с правильным временем.

* Напишите функцию-конструктор new Voter(options) для голосовалки. Она должна получать элемент в options.elem, в следующей разметке:

<div id="voter" class="voter">

<span class="down">—</span>

<span class="vote">0</span>

<span class="up">+</span>

</div>

По клику на + и — число должно увеличиваться или уменьшаться.

Публичный метод voter.setVote(vote) должен устанавливать текущее число – значение голоса.

Все остальные методы и свойства пусть будут приватными.

* Переписать доску для рисования из урока в виде графической компоненты

# Используемая литература

* [Mozilla Developer Network](https://developer.mozilla.org/)
* [MSDN](http://msdn.microsoft.com/)
* [Safari Developer Library](https://developer.apple.com/library/safari/navigation/index.html)
* [Современный учебник JavaScript](http://learn.javascript.ru)