**Современный DOM: полифиллы**

В старых IE, особенно в IE8 и ниже, ряд стандартных DOM-свойств не поддерживаются или поддерживаются плохо.

Если говорить о современных браузерах, то они тоже не все идут «в ногу», всегда какие-то современные возможности реализуются сначала в одном, потом в другом.

Но это не значит, что нужно ориентироваться на самый старый браузер из поддерживаемых!

Для того, чтобы не думать об устаревших браузерах, а писать современный код, который при этом работает везде, используют полифиллы.

**[Полифиллы](http://learn.javascript.ru/dom-polyfill" \l "polifilly)**

«Полифилл» (англ. polyfill) – это библиотека, которая добавляет в старые браузеры поддержку возможностей, которые в современных браузерах являются встроенными.

Один полифилл мы уже видели, когда изучали собственно JavaScript – это библиотека [ES5 shim](https://github.com/es-shims/es5-shim). Если её подключить, то в IE8- начинают работать многие возможности ES5. Работает она через модификацию стандартных объектов и их прототипов. Это типично для полифиллов.

В работе с DOM несовместимостей гораздо больше, как и способов их обхода.

**[Что делает полифилл?](http://learn.javascript.ru/dom-polyfill" \l "chto-delaet-polifill)**

Для примера добавим в DOM поддержку свойства firstElementChild, если её нет. Здесь речь, конечно, об IE8, в других браузерах оно и так поддерживается, но пример типовой.

Вот код для такого полифилла:

if (document.documentElement.firstElementChild === undefined) { // (1)

Object.defineProperty(Element.prototype, 'firstElementChild', { // (2)

get: function() {

var el = this.firstChild;

do {

if (el.nodeType === 1) {

return el;

}

el = el.nextSibling;

} while (el);

return null;

}

});

}

Если этот код запустить, то firstElementChild появится у всех элементов в IE8.

Общий вид этого полифилла довольно типичен. Обычно полифилл состоит из двух частей:

1. Проверка, есть ли встроенная возможность.
2. Эмуляция, если её нет.

**[Проверка встроенного свойства](http://learn.javascript.ru/dom-polyfill" \l "proverka-vstroennogo-svoystva)**

Для проверки встроенной поддержки firstElementChild мы можем просто обратиться к document.documentElement.firstElementChild.

Если DOM-свойство firstElementChild поддерживается, то его значение не может быть undefined. Если детей нет – свойство равно null, но не undefined.

Сравним:

alert( document.head.previousSibling ); // null, поддержка есть

alert( document.head.blabla ); // undefined, поддержки нет

За счёт этого работает проверка в первой строке полифилла.

**Важная тонкость – элемент, который мы тестируем, должен *по стандарту* поддерживать такое свойство.**

Попытаемся, к примеру, проверить «поддержку» свойства value. У input оно есть, у div такого свойства нет:

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

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

alert( input.value ); // пустая строка, поддержка есть

alert( div.value ); // undefined, поддержки нет

**Поддержка значений свойств**

Если мы хотим проверить поддержку не свойства целиком, а некоторых его значений, то ситуация сложнее.

Например, нам интересно, поддерживает ли браузер <input type="range">. То есть, понятно, что свойство type у input, в целом, поддерживается, а вот конкретный тип <input>?

Для этого можно создать <input> с таким type и посмотреть, подействовал ли он.

Например:

<input type="radio">

<input type="no-such-type">

<script>

alert( document.body.children[0].type ); // radio, поддерживается

alert( document.body.children[1].type ); // text, не поддерживается

</script>

1. Первый input имеет type="radio". Этот тип точно поддерживается, поэтому input.type имеет значение "radio", как и указано.
2. Второй input имеет type="no-such-type". В качестве типа, для примера, специально указано заведомо неподдерживаемое значение. При этом input.type равен "text", таково значение по умолчанию. Мы можем прочитать его и увидеть, что поддержки нет.

Эта проверка работает, так как хоть в HTML-атрибут type и можно присвоить любую строку, но DOM-свойство type [по стандарту](http://www.w3.org/TR/html-markup/input.html) хранит реальный тип input'а.

**[Добавляем поддержку свойства](http://learn.javascript.ru/dom-polyfill" \l "dobavlyaem-podderzhku-svoystva)**

Если мы осуществили проверку и видим, что встроенной поддержки нет – полифилл должен её добавить.

Для этого вспомним, что DOM элементы описываются соответствующими JS-классами.

Например:

* <li> – [HTMLLiElement](http://www.w3.org/TR/html5/grouping-content.html" \l "the-li-element)
* <a> – [HTMLAnchorElement](http://www.w3.org/TR/html5/text-level-semantics.html" \l "the-a-element)
* <body> – [HTMLBodyElement](http://www.w3.org/TR/html5/sections.html" \l "the-body-element)

Они наследуют, как мы видели ранее, от [HTMLElement](http://www.w3.org/TR/html5/dom.html" \l "htmlelement), который является общим родительским классом для HTML-элементов.

А HTMLElement, в свою очередь, наследует от [Element](http://www.w3.org/TR/dom/" \l "interface-element), который является общим родителем не только для HTML, но и для других DOM-структур, например для XML и SVG.

**Для добавления нужной возможности берётся правильный класс и модифицируется его prototype.**

Например, можно добавить всем элементам в прототип функцию:

Element.prototype.sayHi = function() {

alert( "Привет от " + this );

}

document.body.sayHi(); // Привет от [object HTMLBodyElement]

Сложнее – добавить свойство, но это тоже возможно, через Object.defineProperty:

Object.defineProperty(Element.prototype, 'lowerTag', {

get: function() {

return this.tagName.toLowerCase();

}

});

alert( document.body.lowerTag ); // body

**Геттер-сеттер и IE8**

В IE8 современные методы для работы со свойствами, такие как [Object.defineProperty](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty), [Object.getOwnPropertyDescriptor](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyDescriptor) и другие не поддерживаются для произвольных объектов, но отлично работают для DOM-элементов.

Чем полифиллы и пользуются, «добавляя» в IE8 многие из современных методов DOM.

**[Какова поддержка свойства?](http://learn.javascript.ru/dom-polyfill" \l "kakova-podderzhka-svoystva)**

А нужен ли вообще полифилл? Какие браузеры поддерживают интересное нам свойство или метод?

Зачастую такая информация есть в справочнике MDN, например для метода remove(): <https://developer.mozilla.org/en-US/docs/Web/API/ChildNode.remove> – табличка совместимости внизу.

Также бывает полезен сервис [http://caniuse.com](http://caniuse.com/), например для elem.matches(css): <http://caniuse.com/#feat=matchesselector>.

**[Итого](http://learn.javascript.ru/dom-polyfill" \l "itogo)**

Если вы поддерживаете устаревшие браузеры – и здесь речь идёт не только про старые IE, другие браузеры тоже обновляются не у всех мгновенно – не обязательно ограничивать себя в использовании современных возможностей.

Многие из них легко полифиллятся добавлением на страницу соответствующих библиотек.

Для поиска полифилла обычно достаточно ввести в поисковике "polyfill", и нужное свойство либо метод. Как правило, полифиллы идут в виде коллекций скриптов.

**Полифиллы хороши тем, что мы просто подключаем их и используем везде современный DOM/JS, а когда старые браузеры окончательно отомрут – просто выкинем полифилл, без изменения кода.**

Типичная схема работы полифилла DOM-свойства или метода:

* Создаётся элемент, который его, в теории, должен поддерживать.
* Соответствующее свойство сравнивается с undefined.
* Если его нет – модифицируется прототип, обычно это Element.prototype – в него дописываются новые геттеры и функции.

Другие полифиллы сделать сложнее. Например, полифилл, который хочет добавить в браузер поддержку элементов вида <input type="range">, может найти все такие элементы на странице и обработать их, меняя внешний вид и работу через JavaScript. Это возможно. Но если уже существующему <input> поменять type на range – полифилл не «подхватит» его автоматически.

Описанная ситуация нормальна. Не всегда полифилл обеспечивает идеальную поддержку наравне с родными свойствами. Но если мы не собираемся так делать, то подобный полифилл вполне подойдёт.

Один из лучших сервисов для полифиллов: [polyfill.io](http://polyfill.io/). Он даёт возможность вставлять на свою страницу скрипт с запросом к сервису, например:

<script src="//cdn.polyfill.io/v1/polyfill.js?features=es6"></script>

При запросе сервис анализирует заголовки, понимает, какая версия какого браузера к нему обратилась и возвращает скрипт-полифилл, добавляющий в браузер возможности, которых там нет. В параметре features можно указать, какие именно возможности нужны, в примере выше это функции стандарта ES6. Подробнее – см. [примеры](http://polyfill.webservices.ft.com/v1/docs/examples) и [список возможностей](http://polyfill.webservices.ft.com/v1/docs/features/).

Также есть и другие коллекции, как правило, полифиллы организованы в виде коллекции, из которой можно как выбрать отдельные свойства и функции, так и подключить всё вместе, пачкой.

Примеры полифиллов:

* <https://github.com/jonathantneal/polyfill> – ES5 вместе с DOM
* <https://github.com/termi/ES5-DOM-SHIM> – ES5 вместе с DOM
* <https://github.com/inexorabletash/polyfill> – ES5+ вместе с DOM

Более мелкие библиотеки, а также коллекции ссылок на них:

* <http://compatibility.shwups-cms.ch/en/polyfills/>
* <http://html5please.com/#polyfill>
* <https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills>

Конечно, можно собрать и свою библиотеку полифиллов самостоятельно из различных коллекций, которые перечислены выше, а при необходимости и написать самому. В этой части учебника мы изучим ещё много методов работы с DOM, которые в этом помогут.