Подготовила:

Якушева Анастасия

ДКО-122б

**18 июня 2015 года одобрена спецификация ES6**

Генеральное собрание Ecma International объявило о ратификации стандарта ECMA-262 6 версии, который является спецификацией языка ECMAScript 6 (ES6), который так же называют ECMAScript 2015.

Официальная страница стандарта:

<http://www.ecma-international.org/publications/standards/Ecma-262.htm>

ES6 – огромный шаг вперед после ES5, спецификация новой версии языка состоит из 600 страниц, ее предшественница – всего из 245. В ES6 добавили **модули** и **классы**; а так же много полезностей, таких как: **Maps**, **Sets**, **Promises** и **Generators**. /\*Ну, и ещё много чего… на 600 станиц в целом\*/

Несмотря на то, что это очень крупный релиз, ES6 обладает полной обратной совместимостью. Комиссия решила не ломать интернет из-за несовместимых версий, поэтому весь старый код будет работать, а переход на новый будет довольно гладким. Это означает, что некоторые проблемы ES5, на которые разработчики жаловались годами так же никуда не денутся.

*Некоторые новшества ES6 (с воспоминаем ES5)*

|  |
| --- |
| **LET**  Все привыкли к объявлению переменных с помощью var. Теперь же можно использовать также и let. Тонкое различие заключается в области видимости. В то время как var в качестве области видимости дает в результате тело функции, в которой объявлена переменная, то область видимости переменной, объявленной с помощью let, – это только блок, в котором выполнено объявление.  Это поможет сделать код более чистым, в результате уменьшения количества висящих переменных в области видимости. Например, классическая итерация массива:    Часто можно было бы использовать, например, переменную j для другой итерации в этой же области видимости. Но с let можно смело объявлять i снова, так как переменные объявлены и доступны только внутри области видимости их собственного блока. |
| **Стрелочные функции**  Стрелочные функции – отличное дополнение к языку JavaScript. Они создают короткий и лаконичный код. Следующий фрагмент кода показывает стрелочную функцию в сравнении аналогичной функцией, написанной в знакомом стиле ES5:    Стрелочные функции означают не только меньшее количество напечатанных символов, но их поведение отличается от обычных функций. Выражение стрелочной функции наследует this и аргументы из окружающего контекста. Это означает, что можно избавиться от уродливых выражений, подобных var that = this, и не нужно связывать функции с правильным контекстом. Пример кода (//обратить внимание на this.title в сравнении с that.title в версии ES5!!!):     |  | | --- | | **Math**  В объект Math была добавлена пара новых методов.   * Math.sign возвращает знак числа как 1, -1 или 0; * Math.trunc возвращает переданное число без цифр после запятой; * Math.cbrt возвращает кубический корень числа.   Больше инфы о математических возможностях ES6: <http://www.2ality.com/2015/04/numbers-math-es6.html> | |
| **Методы** К прототипу String была добавлена пара удобных методов. Большинство из них служат просто устранения обходных путей с методом indexOf() для получения того же результата:    Просто, но эффективно. Другой удобный метод был добавлен для создания повторяющейся строки: |
| **Шаблонные литералы** Шаблонные литералы обеспечивают чистый способ для создания строк и интерполяции выражений. Синтаксис основан на знаке доллара и фигурных скобках ${...}. Шаблонные литералы заключаются в обратные кавычки. Краткий пример:    Как видно из кода выше, шаблонные литералы просто удобны для конкатенации по сравнению с ES5. Однако шаблонные литералы также можно использовать и для многострочных литералов. //Пробельные символы (например, символ новой строки) являются частью строки!!! |
| **Массивы**  Появилось несколько новых статических методов класса у объекта Array и несколько новых методов у прототипаArray.  Во-первых, Array.from создает экземпляры Array из массивоподобных и итерируемых объектов. Примеры массивоподобных объектов включают в себя:   * arguments внутри функции; * nodeList, возвращенный методом document.getElementsByTageName(); * данные новых структур Map и Set.     В приведенном выше примере, можно увидеть, что у массива элементов есть метод forEach, который недоступен в коллекции itemElements.  Интересной особенностью Array.from является второй необязательный аргумент mapFunction, который позволяет создать новый распределяемый массив в одном вызове.    Далее есть Array.of, который ведет себя так же, как конструктор Array. Он исправляет особый случай, когда в качестве аргумента передается одно число. Это делает Array.of более предпочтительным по сравнению с new Array().    Последняя, но не менее важная, пара методов была добавлена для прототипа Array. find возвращает первый элемент, для которого функция обратного вызова вернет true;   * findIndex возвращает индекс первого элемента, для которого функция обратного вызова вернет true; * fill «перезаписывает» элементы массива переданным аргументом. |
| Оператор распространения Оператор распространения (...) – это очень удобный синтаксис для разворачивания элементов массива в определенных местах, например, в качестве аргументов в вызовах функций. Показать несколько примеров – это, вероятно, лучший способ продемонстрировать, насколько они полезны.  Во-первых, как развернуть элементы массива внутри другого массива??    Синтаксис распространения также удобен при вызове функций с аргументами:    Как можно видеть, это спасает от часто используемого обхода fn.apply(). Синтаксис очень гибок, поэтому оператор распространения может использоваться в любом месте в качестве списка аргументов. Это означает, что следующий вызов дает тот же результат:    Оператор распространения можно применять к массивам и аргументам, а также он может быть применен КО ВСЕМ итерируемым объектам, таким как NodeList:    Теперь allTheThings – это одномерный массив, содержащий элементы <form> и их дочерние элементы <input> и<select>. |
| Деструктурирование Деструктурирование обеспечивает удобный способ для извлечения данных из объектов или массивов. Для начала, хороший пример может быть показан на массиве:    С этим синтаксисом нескольким переменным могут быть присвоены значения за один раз. Приятный побочный эффект заключается в том, что можно легко поменять местами значения переменных:    Деструктурирование также работает и с объектами. Убедитесь в наличии соответствующих ключей:    Также вы можно использовать этот механизм, чтобы изменить имена переменных:  Еще один интересный образец имитации нескольких возвращаемых значений:    Деструктурирование может быть использовано, чтобы задать значения по умолчанию для объектов аргументов. С литералом объекта можно имитировать именованные параметры. |
| **Значения по умолчанию** Значения по умолчанию для параметров функций доступны в ES6 со следующим синтаксисом:    А в ES5 было бы примерно так…    Значение по умолчанию для аргумента вызывается либо undefined, либо отсутствием аргумента. |
| МодулиМодули Модули – долгожданное дополнение к языку JavaScript. (//Мб даже главное, ради чего стоит копаться в ES6!)  Сегодня любой серьезный JavaScript проект использует какой-либо тип модульной системы, возможно что-то вроде шаблона «открытый модуль» или более обширные форматы AMD или CommonJS. Тем не менее, браузеры не располагают каким-либо типом модульной системы. Вам всегда необходим либо этап сборки, либо загрузчик ваших модулей AMD или CommonJS. Инструменты для обработки этого включают в себя RequireJS, Browserify и Webpack.  Спецификация ES6 включает в себя как новый синтаксис, так и механизм загрузчика модулей. Если хочется использовать модули и писать на будущее, то нужно использовать данный синтаксис. Современные инструменты сборки поддерживают этот формат, возможно через плагин, поэтому всё должно пройти хорошо (не беспокойтесь, мы обсудим это далее в разделе «Транспиляция»).  А теперь о синтаксисе модулей ES6. Модули разработаны вокруг ключевых слов export и import. Рассмотрим пример с двумя модулями сразу:      Как можно видеть, возможен экспорт нескольких выражений export. Каждое должно явно указывать тип экспортируемого значения (в этом примере function и var).  Выражение import в этом примере использует синтаксис (подобный деструктуризации), чтобы явно определить, что будет импортировано. Чтобы импортировать модуль целиком, должен быть использован символ \* в сочетании с ключевым словом, чтобы дать модулю локальное имя:    В модульной системе есть экспорт default, который может быть функцией. Чтобы импортировать это значение по умолчанию в модуле, необходимо только предоставить локальное имя (т.е. без деструктурирования):    //!!! выражения import синхронны, но код модуля не выполняется, пока все зависимости не загрузятся. |
| Классы Классы – широко обсуждаемая особенность ES6. Некоторые считают, что они идут против природы прототипов JavaScript, в то время как другие думают, что они понижают входной барьер для начинающих и людей, приходящих из других языков, и что они помогают людям писать большие приложения. В любом случае, они являются частью ES6. Далее приведено очень краткое введение в классы JavaScript.  Классы построены вокруг ключевых слов class и constructor. Небольшой пример:    //!!!Обратить внимание, что определение класса не является обычным объектом, следовательно, нет запятых между членами класса!  Чтобы создать экземпляр класса, вы нужно использовать ключевое слово new. Для наследования базового класса, использовать extend:    Из любого конструктора или метода производного класса можно использовать super, чтобы получить доступ к базовому классу:   * чтобы вызвать родительский конструктор, использовать super(); * чтобы вызвать другой метод, использовать, например, super.getName().   Более подробно о классах: <http://www.2ality.com/2015/02/es6-classes-final.html> |
| Символы Символы – новый примитивный тип данных, как, например, Number и String. Можно использовать символы, чтобы создать уникальные идентификаторы для свойств объекта, или чтобы создать уникальные константы.    //!!!Обратить внимание, что набор пар ключ-значение с символами не возвращается методом Object.getOwnPropertyNames(), и они не видны в итерациях for...in, Object.keys() или JSON.stringify(). Этим они отличаются от обычных ключей на основе строк.  Можете получить массив символов объекта с помощью Object.getOwnPropertySymbols().  Символы, естественно, работают с const из-за своего неизменяемого характера:    Можно добавить описание к символу. Его нельзя использовать для получения доступа к самому символу, но оно может быть полезно при отладке:    Хотите узнать больше о символах? На Mozilla Developer Network есть [хорошая статья о них](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol). |

**Совместимость с браузерами**





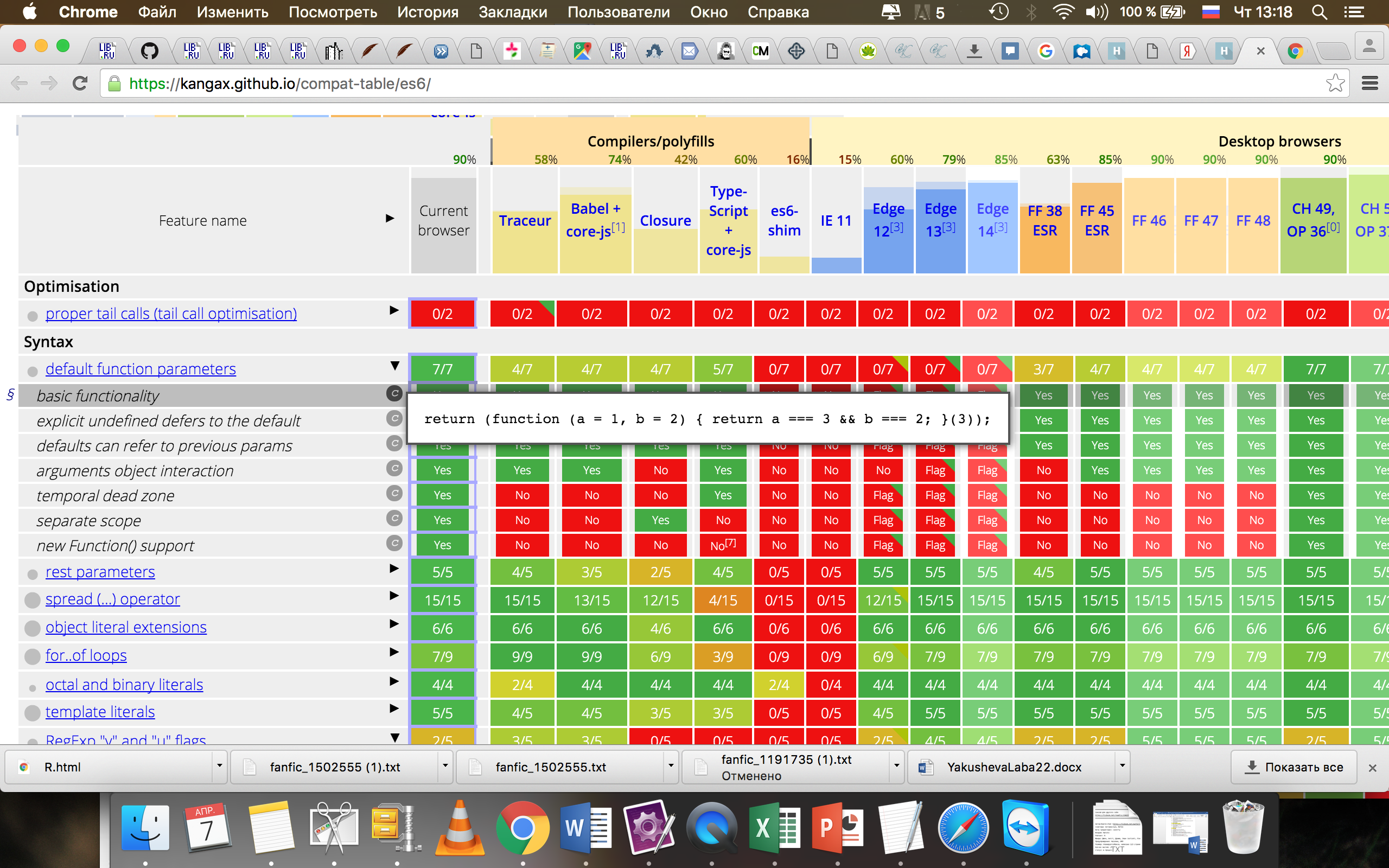
Однако, чтобы получить наиболее актуальную информацию по поддержке ES6 тем или иным браузером удобнее всего получить по ссылке <https://kangax.github.io/compat-table/es6/>, которую должно открыть теми браузерами, которые вас инетересуют.

Лично на моем ноутбуке стоит 2 браузера. Google Chrome (Версия 49.0.2623.110 (64-bit)) и Safari (Версия 9.0.1 (11601.2.7.2)).

Ниже приведена малая часть таблицы для Google Chrome (слева) и Safari (справа) ранее названных версий.



Также стоит отметить, что с помощью этой же таблицы довольно удобно ознакомиться и непосредственно с «нововведениями», их синтаксисом. Скрин ниже сделан с браузера Google Chrome (версии 49.0.2623.110 (64-bit)) и иллюстрирует пример «Значения параметров по умолчанию», базовый функционал. Который, к слову, поддерживается в Google Chrome данной версии, но не поддерживается Safari (версии 9.0.1 (11601.2.7.2)).



Кроме того на этом же сайте можно ознакомиться и с тем, какие нововведения планируются в следующих версиях ECMAScript. И даже посмотреть, что уже на данный момент поддерживается тем или иным браузером.   
  
Ниже представлена часть таблицы для Google Chrome (версии 49.0.2623.110 (64-bit)).



**Полезные ссылки:**

* [Описание нововведений](http://es6-features.org/) (на языке оригинала… Но разобраться вполне реально)
* [Транслятор в более ранние версии языка](https://babeljs.io/repl/) (комментарии излишни)
* [Таблица совместимости с браузерами](https://kangax.github.io/compat-table/es6/) (да-да, та самая, которая рассматривалась выше)