Привет всем, меня зовут Ян, и сегодня я бы хотел поговорить о такой важной теме для новичков в программировании, как стандарт ECMAScript.

ECMAScript 2015, также известный как ES6, является фундаментальной версией стандарта ECMAScript.

Поскольку длительное время прошло между ES5.1 и ES6, релиз содержит важные новые функции и значительные изменения в предлагаемых передовых методах разработки программ JavaScript. Чтобы понять, насколько фундаментальным является ES2015, просто имейте в виду, что с этой версией спецификационный документ переходил от 250 страниц до ~ 600.

Наиболее важные изменения в ES2015 включают:

1. let, const и блочная область видимости

2. Стрелочные функции

3. Параметры по умолчанию

4. Spread / Rest оператор

5. Расширение возможностей литералов объекта

6. Восьмеричный и двоичный литералы

7. Деструктуризация массивов и объектов

8. Ключевое слово super для объектов

9. Строковые шаблоны и разделители

10. for...of против for...in

11. Map и WeakMap

12. Set и WeakSet

13. Классы в ES6

14. Тип данных Symbol

15. Итераторы

16. Генераторы

17. Промисы

1. let, const и блочная область видимости

Ключевое слово let позволяет объявлять переменные с ограниченной областью видимости — только для блока {...}, в котором происходит объявление. Это называется блочной областью видимости.

Другой формой объявления переменной с блочной областью видимости является ключевое слово const. Оно предназначено для объявления переменных (констант), значения которых доступны только для чтения. Это означает не то, что значение константы неизменно, а то, что идентификатор переменной не может быть переприсвоен.

О чём стоит помнить:

• Когда дело касается поднятия переменных (hoisting) let и const, их поведение отличается от традиционного поведения var и function. И let и const не существуют до своего объявления

• Областью видимости let и const является ближайший блок.

• При использовании const рекомендуется использовать ПРОПИСНЫЕ\_БУКВЫ.

• В const одновременно с объявлением переменной должно быть присвоено значение.

2. Стрелочные функции

Стрелочные функции представляют собой сокращённую запись функций в ES6. Стрелочная функция состоит из списка параметров ( ... ), за которым следует знак => и тело функции.

Это ещё не всё!...

Стрелочные функции не просто делают код короче. Они тесно связаны с ключевым словом this и привязкой контекста.

Поведение стрелочных функций с ключевым словом this отличается от поведения обычных функций с this. Каждая функция в JavaScript определяет свой собственный контекст this, но внутри стрелочных функций значение this то же самое, что и снаружи (стрелочные функции не имеют своего this).

3. Параметры по умолчанию

ES6 позволяет установить параметры по умолчанию при объявлении функции. Вот простой пример:

4. Spread / Rest оператор

... оператор называют как spread или rest, в зависимости от того, как и где он используется.

При использовании в любом итерируемом объекте (iterable), данный оператор "разбивает" ("spread") его на индивидуальные элементы:

Другим распространённым использованием оператора ... является объединение набора значений в один массив. В данном случае оператор работает как "rest"

5. Расширение возможностей литералов объекта

ES6 позволяет объявить литералы объекта с помощью короткого синтаксиса для инициализации свойств из переменных и определения функциональных методов. Также, стандарт обеспечивает возможность вычисления свойств непосредственно в литерале объекта.

6. Восьмеричный и двоичный литералы

В ES6 появилась новая поддержка для восьмеричных и двоичных литералов.

7. Деструктуризация массивов и объектов

Деструктуризация помогает избежать использования вспомогательных переменных при взаимодействии с объектами и массивами.

8. Ключевое слово super для объектов

ES6 позволяет использовать метод super в (безклассовых) объектах с прототипами.

9. Строковые шаблоны и разделители

ES6 предоставяляет более простой способ вставки значения переменной или результата выражения (т.н. "интерполяцию"), которые рассчитываются автоматически.

• ${ ... } используется для вычисления значения переменной/выражения.

• `` Обратные кавычки используются как разделитель для таких случаев.

10. for...of против for...in

• for...of используется для перебора в цикле итерируемых объектов, например, массивов.

• for...in используется для перебора в цикле всех доступных для перебора (enumerable) свойств объекта.

11. Map и WeakMap

ES6 представляет новые структуры данных — Map и WeakMap. На самом деле, мы используем "Map" в JavaScript всё время. Каждый объект можно представить как частный случай Map.

Классический объект состоит из ключей (всегда в строковом виде) и значений, тогда как в Map для ключа и значения можно использовать любое значение (и объекты, и примитивы)

WeakMap

WeakMap это Map, в котором ключи обладают неустойчивыми связями, что позволяет не мешать сборщику мусора удалять элементы WeakMap. Это означает, что можно не беспокоиться об утечках памяти.

Стоить отметить, что в WeakMap, в отличие от Map, каждый ключ должен быть объектом.

Для WeakMap есть только четыре метода: delete(ключ), has(ключ), get(ключ) и set(ключ, значение).

12. Set и WeakSet

Объекты Set это коллекции уникальных значений. Дублированные значения игнорируются, т.к. коллекция должна содержать только уникальные значения. Значения могут быть примитивами или ссылками на объекты.

WeakSet

Аналогично WeakMap, объект WeakSet позволяет хранить объекты с неустойчивыми связями в коллекции. Объект в WeakSet уникален.

13. Классы в ES6

В ES6 представили новый синтаксис для классов. Здесь стоит отметить, что класс ES6 не представляет собой новую объектно-ориентированную модель наследования. Это просто синтаксический сахар для существующего в JavaScript прототипного наследования.

Класс в ES6 представляет собой просто новый синтаксис для работы с прототипами и функциями-конструкторами, которые мы привыкли использовать в ES5.

extends и super в классах

В ES6 ключевое слово extends позволяет классу-потомку наследовать от родительского класса. Важно отметить, что конструктор класса-потомка должен вызывать super().

Также, в классе-потомке можно вызвать метод родительского класса с помощью super.parentMethodName()

О чём стоит помнить:

• Объявления классов не поднимаются наверх (not hoisted). Сначала нужно объявить класс и только после этого использовать его, иначе будет ошибка ReferenceError.

• Нет необходимости использовать ключевое слово function во время задания функций внутри определения класса.

14. Тип данных Symbol

Symbol это уникальный и неизменяемый тип данных, представленный в ES6. Целью Symbol является создание уникального идентификатора, к которому нельзя получить доступ.

Если Symbol используется как свойство/ключ объекта, он сохраняется таким специальным образом, что свойство не будет показано при нормальном перечислении свойств объекта.

Чтобы извлечь символьные свойства объекта, нужно использовать Object.getOwnPropertySymbols()

15. Итераторы

Итератор обращается к элементам коллекции по одному, в то же время сохраняя память о своей текущей позиции в этой коллекции. У итератора есть метод next(), который возвращает следующий элемент в последовательности. Этот метод возвращает объект с двумя свойствами: done (окончен ли перебор) и value (значение).

В ES6 есть метод Symbol.iterator, который определяет итератор для объекта по-умолчанию. При каждой необходимости перебора в цикле для объекта (например, в начале цикла for..of), его метод итератора вызывается без аргументов, и возвращённый итератор используется для того, чтобы получить значения для перебора.

16. Генераторы

Функции-генераторы представляют собой новую особенность ES6, которая позволяет функции создавать много значений в течение некоторого периода времени, возвращая объект (называемый генератором), который может быть итерирован для выброса значений из функции по одному за раз.

Функция-генератор возвращает итерируемый объект при своём вызове.

Функция-генератор записывается с помощью знака \* после ключевого слова function, а в теле функции должно присутствовать ключевое слово yield.

17. Промисы

В ES6 появилась встроенная поддержка промисов. Промис это объект, который ждёт выполнения асинхронной операции, после которого (т.е. после выполнения) промис принимает одно из двух состояний: fulfilled (resolved, успешное выполнение) или rejected (выполнено с ошибкой).

Стандартным способом создания промиса является конструктор new Promise(), который принимает обработчик с двумя функциями как параметрами. Первый обработчик (обычно именуемый resolve) представляет собой функцию для вызова вместе с будущим значением, когда оно будет готово; второй обработчик (обычно именуемый reject) является функцией, которая вызывается для отказа от выполнения промиса, если он не может определить будущее значение.

Каждый промис обладает методом then, в котором есть два коллбэка. Первый коллбэк вызывается, если промис успешно выполнен (resolved), тогда как второй коллбэк вызывается, если промис выполнен с ошибкой (rejected).

При возвращении промиса, успешно обработанное значение промиса пройдёт к следующему коллбэку, для того, чтобы эффективно соединить их вместе.

Эта простая техника помогает избежать ада с коллбэками ("callback hell").

При возвращении значения от then коллбэки передадут значение следующему коллбэку then.

Это было краткое описание всех новых функций, появившихся при выпуске ES6. Давайте учиться и использовать эти функции правильно. Спасибо и удачи всем.