ES6

Next version of JavaScript language

Maha - Technical Lead

Introduction

- JavaScript was created within 10 days in May 1995
- First official release was in June 1997
- Editorial changes to specification in 2nd version
- 3rd version released in December 1999 with
  - Regular expressions, better string handling, new control statements, try/catch exception handling
- 4th Version was abandoned due to political differences concerning language complexity
- 5th version released December 2009 with
  - Added strict mode, getters and setters
  - Library support for JSON
- 6th version was finalized in June 2015

New features

- Block Binding
  - let and const
- Arrow functions
- Default + rest + spread
- Destructuring assignment
- Enhanced Object Literals
- Template Strings
- Classes
- Symbols
- For-of, Iterators and Generators

New features contd..

- Collections
  - Map and WeakMap
  - Set and WeakSet
- Promises
- Modules
- Tail Call Optimization
- Proxies and Reflect api
- Addition to existing APIs
  - Number, String, Object and Array
- Sub Classable built-ins

Block Binding- let and const

- let variables are block-scoped. The scope of a variable declared with let is just the enclosing block, not the whole enclosing function.
- Loops of the form for (let x...) create a fresh binding for x in each iteration.
- Global let variables are not properties on the global object
- It’s an error try to use a let variable before its declaration is reached. This is called as Temporal dead zone(TDZ)
- Redeclaring a variable with let is a SyntaxError
- const is similar to let but creates immutable variables. Can't re assign values
- declaration of const requires value assignment. SyntaxError otherwise

Arrow functions

- Simplified form of function declaration
- Lexical this, arguments, super and new.target
- Although it is a function, is not a constructor
- It has no prototype
- Can not be used as generators

Default + Rest + Spread

- Default - parameters allow formal parameters to be initialized with default values if no value or undefined is passed
- The default parameters gets evaluated at call time, we can use any variable, function and other formal parameter but not any variable inside the function
- Rest - The rest parameters syntax enables you to pull a real Array out of the function‘s arguments by adding a parameter name prefixed by …
- Rest parameter must be last formal parameter of the function
- The spread operator allows an expression to be expanded in places where multiple arguments or multiple elements are expected. Below are advantages
  - A better apply
  - A more powerful array literal
  - Apply for new

Destructuring assignment

- Destructuring assignment allows you to assign the properties of an array or object to variables using syntax that looks similar to array or object literals
- Helps to return multiple values from function
- With destructuring assignment, we can avoid boilerplate code and code becomes more readable

Enhanced Object Literals

- Simplified version of declaring object literals
- Property value shorthand, not required to use : if property name is same as variable name
- Computed property keys - We can use dynamically computed property names
- Method definition shorthand - colon (:) and the function keyword are optional
- Define prototypal inheritance through __proto__
- Duplicates properties allowed even in strict mode but the second property will overwrite the first.

Template Strings

- Template strings are regular strings enclosed by the backtick ` character instead of double or single quotes.
- Multiline strings - A formal way of creating multiline strings.
- Expression interpolation or string formatting - The ability to substitute parts of the string for values of variables or any JS expressions.
- Tagged template strings - you can modify the output of template strings using a function.
- String.raw method is a tag function, can be used to get the raw string form of template strings (that is, the original, uninterpreted text).

Classes

- Classes provide a much simpler and clearer syntax to create objects and deal with inheritance. They are similar to functions except
  - Class declarations are not hoisted. Need to declare before using it
  - All code inside of classes runs in strict mode automatically. There’s no way to opt-out
  - Calling the class constructor without new throws an error
  - All methods are non-enumerable
- The constructor method is a special method for creating and initializing an object created with a class. Create all properties inside constructor method
- We can define static members using keyword static. Static members can be used without creating instance of a class.
- We can have class expressions similar to function expressions
- Classes are also first class citizens

Classes and inheritance

- Use keyword extends to inherit from other class
- You can use extends with any expression as long as the expression resolves to a function with [[Construct]] and a prototype.
- Super can only be used inside class or function which is extended from another class/function, Throws error otherwise.
- Must call super() before accessing this in the constructor.
- extends also inherits static members

Symbols

- A symbol is a unique and immutable data type and can be used as an identifier for object properties. Not equal to any other symbol except itself
- Symbol keys are designed to avoid collisions, so keys won't appear in for-in, Object.keys() and Object.getOwnPropertyNames()
- Keys not for exactly private. All keys can be accessed using Object.getOwnPropertySymbols(obj)
- There are three ways to obtain a symbol
  - Call Symbol() returns a new unique symbol each time it's called.
  - Call Symbol.for(string). for accessing set of existing symbols in symbol registry.Can be useful when needed shared symbol across different modules and pages
  - Use symbols like Symbol.iterator, defined by the standard. A few symbols are defined by the standard itself. Each one has its own special purpose. Used for backward compatibility

For-of

- New way of loop over array elements, direct syntax, avoids pitfall of for-in and unlike forEach, it can break, continue and return from loop
- For-in loops over object properties and for-of loops over object data
- It can loop over strings, collections, iterable objects, generators and DOM objects
- For-of statement will throw an error when used on a non-iterable, null, or undefined

Iterators

- Iterator design pattern is used to access the elements of a collection without knowing its underlying representation.
- Iterable object should have method [Symbol.iterator] and should return an object containing next method. The next method returns a result object. The result object has two properties, value, which is the next value, and done. true when done=true there are no more values to return.
- Iterator gives the way to iterate over any user defined data types

How does for-of work

- A for–of loop starts by calling the [Symbol.iterator](), This returns a new iterator object. This object will be having next method
- For–of loop will call next method repeatedly on iterator, once each time through the loop.
- next method return object containing {done:<true or false>, value:value}
- Loop stops when done is true

Generators

- Regular functions start with function. Generator-functions start with function*
- Regular function use keyword return, can only return once but generator-function use keyword yield, can yield any number of times.
- Regular functions can’t pause themselves. Generator-functions can.
- Generators are iterators
- Object with next method implicitly created
- yield ‘value’ implicitly create result object as {value:’value’, done:false}
- yield statement interrupts execution while remembering the state.
- Generators can be used with spread operators

Collections

- Javascript introduces 4 collections
  - Set and WeakSet - Stores unique elements.
  - Map and WeakMap - Stores Key and value where key can be any javascript value
- Set Operations
  - new set, new set(iterable) - creates new set
  - set.size gets the number of values in the set.
  - set.has(value) returns true if the set contains the given value.
  - set.add(value) adds a value to the set. If the value was already in the set, nothing happens.
  - set.delete(value) removes a value from the set.
  - set.clear() removes all values from the set.
- Set is iterable, support both for-of and forEach
- Equality is based on Same-value-zero algorithm

Collections - Map

- A Map is a collection of key-value pairs. Here's what Map can do
  - new Map or new Map(pairs) creates a new map
  - map.size gets the number of entries in the map.
  - map.has(key) tests whether a key is present.
  - map.get(key) gets the value associated with a key, or undefined if there is no such entry
  - map.set(key, value) adds or overwrites an entry to the map associating key with value
  - map.delete(key) deletes an entry.
  - map.clear() removes all entries from the map.
  - map.keys() returns an iterator over all the keys in the map.
  - map.values() returns an iterator over all the values in the map.
  - map.entries() returns an iterator over all the entries in the map
- Map is iterable. support both for-of and forEach
- Equality is based on Same-value-zero algorithm

Collections - WeakSet and WeakMap

- They are similar to Set and Map but with a few restrictions:
  - WeakMap supports only new, .has(), .get(), .set(), and .delete().
  - WeakSet supports only new, .has(), .add(), and .delete().
- The values stored in a WeakSet and the keys stored in a WeakMap must be objects.
- Both type of weak collection is not iterable.
- WeakSet and Key in WeakMap does not keep a strong reference to the objects it contains. When an object in a WeakSet is garbage collected, it is simply removed from the collection

Promises

- The Promise object is used for deferred and asynchronous operations. Represents an operation that hasn't completed yet, but in future.
- Lifecycle - Pending, Fulfilled or Rejected
- new Promise(function(resolve, reject)) - Creates a new promise and immediately run function passed. resolve and reject are callable functions, will be invoked asynchronously. resolve(result) will mark the promise as resolved and reject(cause) will mark the promise as failed or rejected.
- promise.then(onResolve, onReject) - onResolve(result) called when success and onReject(cause) called when failed
- promise.catch(onReject) - is same as promise.then(null, onReject)
- Any return statement inside promise.then always returns a promise

Promises Contd..

- We can chain promises
- Promise.resolve(result) method returns a Promise object that is resolved with the given value. static utility for creating a promise
- Promise.reject(cause) Returns a Promise object that is rejected with the given reason.
- Promise.all([promise1, promise2, …]) It returns a promise which resolves when all promises have resolved, result is an array containing the results of every promise, or is rejected when any of promises are rejected. with an cause from first rejection
- Promise.race([p1, p2..]) method returns a promise that resolves or rejects as soon as one of the promise resolves or rejects, with the value or reason from that promise.

Modules

- Module is a file containing JS code. Provides ability to split code into multiple manageable files using import and export keyword
- Module code automatically runs in strict mode and there’s no way to opt-out
- Variables created in module are exist only within scope of the module. export anything that should be available to code outside of the module.
- Use import keyword to import code from other modules
- import and export are allowed only at top level in a module. There are no conditional imports or exports
- There can be only one default export
- Modules are singletons. Even if a module is imported multiple times, only a single “instance” of it exists

Tail call optimization

- Tail call optimization allows some function calls to be optimized in order to keep a smaller call stack. instead of creating a new stack frame for each call, the current stack frame will be cleared and reused
- Tail call optimization happens in below cases
  - The last thing to evaluate before the return statement is a function call
  - The function making the tail call has no further work to do after the tail call returns
  - Tail call does not require access to local variables in the current stack frame
- Tail call optimization Will be helpful in recursive calls, use less memory, and prevent stack overflow errors.

Proxies and Reflect API

- The Proxy object is used to define custom behavior for fundamental operations
- Concepts
  - Handler - Placeholder object which contains traps.
  - Traps - The methods that provide property access.
  - Target - Object which the proxy virtualizes. It is often used as storage backend for the proxy.
- Useful in observing, logging, validating and restricting accesses to an object
- Reflect is a new built-in object that provides methods for interceptable JavaScript operations. Reflect methods are the same as those of proxy handlers.

Number Changes

- Binary and Octal Literals
  - 0b11 or 0B11 is 3 and 0b100 is 4
  - 0o10 or 0O10 is 8
- Number.isNaN and Number.isFinite are like their global namesakes, except that they don’t coerce input to Number
- Number.parseInt and Number.parseFloat are same as their global namesakes
- Number.isInteger checks if input is a Number value that doesn’t have a decimal part
- Number.EPSILON helps figure out negligible differences between two numbers – e.g. 0.1 + 0.2 and 0.3
- Number.MAX_SAFE_INTEGER is the largest integer that can be safely represented in JavaScript
- Number.MIN_SAFE_INTEGER is the smallest integer that can be safely represented in JavaScript
- Number.isSafeInteger - checks whether an integer is within safe bounds

String Changes

- String Manipulation, Prototype methods
  - startsWith - returns true if string starts with given string
    - “helloworld”.startsWith(“hello”)
    - “helloworld”.startsWith(“world”, 5)
  - endsWith - returns true if string ends with given string
    - “helloworld”.endsWith(“world”)
    - “helloworld”.endsWith(“hello”, 5)
  - includes - returns true if string contains given string
    - “helloworld”.includes(“world”) -> true,
    - “helloworld”.includes(“ggggg”) -> false
  - repeat - Repeats the strings mentioned times. Number should be positive finite number
    - “abc”.repeat(2) ->abcabc,
    - “abc”.repeat(‘dd’)-> ‘’
  - String.prototype[Symbol.iterator]

Object Changes

- Object.assign - copy the values of all enumerable own properties from one or more source objects to a target object. It will return the target object.
  - Object.assign({},{a:1}) -> {a:1}
  - Object.assign({a:1}, {b:2}, {c:3}); -> { a: 1, b: 2, c: 3 }
- Object.is() determines whether two values are the same value
  - Object.is('foo', 'foo'); //true
  - Object.is({a:1}, {a:1}); //false
- Object.getOwnPropertySymbols - As covered in symbols sections, list all symbols in objects
- Object.setPrototypeOf - sets the prototype of a specified object to another object. Replacement for __proto__

Array Changes

- Static methods
  - Array.from - creates a new Array instance from an array-like or iterable object
  - Array.of - method creates a new Array instance with a variable number of arguments, Array.of(1, 2, 3); // [1, 2, 3]
- Prototype Methods
  - .fill - replaces elements with given value. [1, 2, 3].fill(4); // [4, 4, 4]
  - .find - returns a value in the array, if an element satisfies the provided testing function
  - .findIndex - Same as find, but return index instead of value
  - .copyWithin - copies the sequence of array elements within the array
- Related to iterators
  - [Symbol.iterator]
  - .keys, .values and .entries

Sub Classable built-ins

- We can subclass all built-in constructors.
- own exception classes
- class MyError extends Error {
  }
  throw new MyError('Something went wrong!');
- Create subclasses of Array
- class MyArray extends Array {
      constructor(len) {
          super(len);
      }
  }
-

Can I use ES6 Today?

- ES6 browser support - https://kangax.github.io/compat-table/es6/
- No - If we dependant on browsers
- Use transpilers to convert ES6 code into ES5
  - babel and traceur
- Sample todo app with es6 and babel https://github.com/mahalingaiahhr/todo-with-es6

References

- https://leanpub.com/understandinges6/read
- http://exploringjs.com/es6/
- https://ponyfoo.com/articles/es6
- https://hacks.mozilla.org/category/es6-in-depth/
- https://babeljs.io/docs/learn-es2015/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript/ECMAScript_6_support_in_Mozilla
- Examples in this session https://github.com/mahalingaiahhr/es6-examples

Q & A

Feedback

Mahalingaiah.hr@happiestminds.com