ES.next

Amazing New Features In JavaScript

Eng. Niveen Nasr El-Den
SD & Gaming CoE
iTi

Day 2

Amazing New Features In JavaScript

ES6 Default parameter

```
// ES6
function dosomething (x = “nothing was sent“)
{
    //x = typeof x !== “undefined” ? x : ”nothing was sent”
    //x = x || “nothing was sent“;
    console.log (“value is :” + x);
}
```

```
dosomething(“hello”)
// value is : hello

dosomething()
// value is : nothing was sent
```

Option Object

- The options object is a widely used pattern that allows user-defined settings to be passed to a function in the form of properties on an object
- It contains a set of named parameters that are passed into a function
- For a function with four or more arguments it's usually a good idea
- Options objects also make it easy to make parameters optional. When an optional parameter isn't passed in, a default value should be used instead.

Option Object Default Parameter

```
function newDefault(msg = "hello!!", options = {}) {
    let title = options.title;
    let fname = options.fname;
    let lname = options.lname;

    return msg + " " + title + ": " + fname + " " +
lname;
}
```

```
newDefault("morning") //
newDefault("morning", {fname: "ali"}) //
```

Using Named Parameters

- Using named parameters for optional settings makes it easier to understand how a function should be invoked.
- It's okay to omit some options when invoking a function with named parameters.
- It's NOT okay to omit the options argument altogether when invoking a function with named parameters when no default value is set for them.

```
function newDefault(msg = "hello!!", { title, fname, lname} = {}) {
    return msg + " " + title + ": " + fname + " " + lname;
}
```

```
newDefault("morning") //??
newDefault("morning", {fname: "ali"}) //??
```

Object.assign()

- We want to merge *options* and *defaults*. Upon duplicate properties, those from *options* must override properties from *defaults*.
- The *Object.assign* method copies properties from one or more source objects to a target object specified as the very first argument.

```
function newDefault(msg = "hello!!", options = {}) {
    let defaultObj = {
        title: "eng",
        fname: "aaa",
        lname: "bbb"
    };
    let settings = Object.assign({}, defaultObj, options)
    return msg + " " + settings.title + ": " + settings.fname + " " + settings.lname;
}

newDefault("morning", {fname: "ali"}) //"morning eng: ali bbb"
```

Set

- Set is an object that allows storing **unique** values either primitive or objects
- Methods:
  - .add(val)
  - .delete(val)
  - .clear()
  - .has(val)
- Iterator Methods:
  - .entries()
  - .values()
  - .keys()
- Property:
  - .size

```
var mySet = new Set([1,2,"my",9,"sss"]);

var s= new Set();
s.add(1);
```

https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Set#Methods

Map

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map

- Map is an object of key/value pairs
- Both key and value can be either primitive or object values
- Methods:
  - .set(key,val)
  - .get(key)
  - .delete(key)
  - .clear()
  - .has(key)
- Iterator Methods:
  - .entries()
  - .values()
  - .keys()
- Property:
  - .size

```
var myMap = new Map([["a",1],[2,10],["my",9]]);

var m= new Map();
m.set("a",1);
```

for..of

- The for...of statement iterates over property values, and it's a better way to loop over arrays and other iterable objects e.g. String, Set etc.

- The for...of statement cannot be used to iterate over properties in plain JavaScript objects since they are not iterables

```
var fruits = ["apple", "strawberry", "banana"]

for( f in fruits){
    console.log(fruits[f])}

for( f of fruits){
    console.log(f)}
```

Iterables

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols

- Arrays are iterables while plain JavaScript objects aren't
- Iterables must have [Symbol.iterator] method; named @@iterator method.
- Iterables return special object called Iterator.

Iterable

- Iterable object can use
  - for..of
  - destructuring
  - …spread operator
- String, Array, Map, Set,… etc. are iterable objects
- We can create iterable objects using generators

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol/iterator

Iterators

- Iterator object has .next() that returns an object of
  - done property
    - *false*: if the iterator is able to return a value from the collection
    - *true*: if the iterator is past the end of the collection
  - value property
    - value returned by the iterator.
    - When *done* is *true*, this returns *undefined*.

Iterables & Iterators

- An iterable must be an object with a function iterator whose key is Symbol.iterator.
- An Iterator knows how to access items at the collection one at a time and keep track of its current position within the sequence.
- We can create iterable objects either by
  - Implementing @@iterator method or
  - using generators function

Generators

- A generator is a special type of function that works as a factory for iterators.
- It uses function *() that contains one or more yield expression
  - Yield return iterator object
- Generators return an object with next()

```
function * genfn(){
    yield 1;
    yield 2;
    yield 3;
}
```

Symbol Data Type

- Symbol is a new primitive datatype what was previously implemented as internal language behavior
  - it has no literal syntax for creation
  - It is a unique and immutable
  - never clashes with any other property key
  - considered as UUID or GUID
  - can be used as Object key
    - i.e. identifier for object property
  - created via a factory method

Symbol

- Syntax:

Symbol(*[descr.]*)

Symbol.for(*[descr.]*)

discr. is a description of the symbol which can be used for debugging but not to access the symbol itself, shown when printing the symbol

```
var mySym = Symbol("my")
typeof mySym;//symbol
String(mySym)//??
mySym.description
```

Symbol

- JavaScript has some built-in symbols which represent internal language behaviors; these were not exposed to developers before ES2015 like
  - Iteration symbols
    - Symbol.iterator → used by for..of
  - Regular expression symbols properties
    - Symbol.match → used by String.prototype.match
    - Symbol.replace → used by String.prototype.replace
    - etc..
  - etc..

Symbol

- Example:

```
var mySym = Symbol("my")

typeof mySym;//symbol

String(mySym)//??

mySym.toString()//??
```

```
var mySym = Symbol("my")
var mySym2 = Symbol("my")

mySym == mySym2//??
```

```
var mySym = Symbol.for("my")
var mySym2 = Symbol.for("my")

mySym == mySym2//??

Symbol.keyFor(mySym)//??
Symbol.keyFor(mySym2)//??
```

Symbol

```
let sym = Symbol();
let obj = { [sym]: "value" ,
            [Symbol('abc')]:101,
            [Symbol.for('abc')]:"test"
            /*Symbol('x'):'abc'*///error
          };

console.log(obj[sym]);

Object.getOwnPropertySymbols(obj)
obj[(Object.getOwnPropertySymbols(obj))[0]]
obj[(Object.getOwnPropertySymbols(obj))[1]]
obj[(Object.getOwnPropertySymbols(obj))[2]]
```

Symbol

- When a symbol value is used as the identifier in a property assignment,
  - the property (like the symbol) is anonymous; and also is non-enumerable, hence, it will not show up as a member in the loop construct

- Symbol-keyed properties will be completely ignored when using JSON.stringify():

ES7 new Features

- Array.prototype.includes

- Exponential Operator **
  - Example:
    2**5 ➔ Math.pow(2,5)

ES8 new Features

- Object.values(obj)
- Object.entries(obj)
- Object.getOwnPropertyDescriptor(obj,prop)
- Object.getOwnPropertyDescriptors(ctor)

- String Padding
  - .padStart(num)
  - .padEnd(num)

- Trailing Commas in function parameter lists and calls

References

- http://es6-features.org/#Constants
- https://github.com/ericdouglas/ES6-Learning
- http://exploringjs.com/es6/
- http://www.2ality.com/2015/02/es6-classes-final.html
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/static
- https://googlechrome.github.io/samples/classes-es6/

Assignments