前端網絡開發人員課程
(二) 進階網絡程式設計

11. ES6 JS VII: Extensions

Presented by Krystal Institute

erb 僱員再培訓局 Employees Retraining Board
SCOPE School of Continuing and Professional Education 專業進修學院
KRYSTAL INSTITUTE DIGITAL ECONOMY CORE TECHNOLOGY

Learning Objective

- Understand extension methods for arrays, objects and strings
- Recap for every lessons about JS

Content

11.1 Revise on the previous lesson

Template Literals

- Before ES6, strings can be wrapped with single quotes or double quotes
- More code is needed for more string functionality
- Template Literals uses backticks (`) to the wrap the string
- Template Literals can wrap multiline strings, and will record the new lines inside the string
- Single and Double Quotes can also be used without escaping

Template Literals

- Before ES6, passing variables into strings requires the use of operators, which can make a simple sentence lengthy
- Template Literals allows variables to be passed inside the string, allowing more flexibility and shorter code
- It has a special block syntax inside the string ${variable}

Array Destructuring

- Array destructuring provides a much quicker way for assigning variables to items in an iterable object
- Before ES6, indexes will have to be used for each item in the array, which is time consuming and it leads to lots of code

```
function getAge(...ages) {
    return ages;
}

let [x, y, z] = getAge(10,20,30);
console.log(x); //10
```

Array Destructuring

- If the function returns less array items than there are variables, the remaining variables will be undefined
- If the function returns more array items than there are variables, all the variables will be assigned in order, leaving the rest of the array items untouched
- The rest syntax (...) and the Default parameter can be used to assign default values for variables if they become undefined / if they have too many array items

Array Destructuring

- Array destructuring can handle nested arrays in two ways:
- Assign a variable for the whole nested array
- Or assign a variable for each item of the nested array
- Array destructuring can be used to swap values, making this task a lot simpler than before ES6

Arrow Function

- Arrow functions speeds up coding process and shortens code
- By using a arrow prefix =>, it can represent a function block within one line
- Brackets for the variable needed to be included unless it is the only variable used in the body of the function
- Line breaks are not allowed if the arrow is used after the line break

ES6 Modules

- Modules are extensions of the JS Script, allowing you to export functions and variables from other scripts to your own

```
    <div id="display"></div>
<script>
    // Assuming this is on the message.js file
    export let message = "This is a message"

    import { message } from './message.js'
    let div = document.querySelector("#display");
    div.textContent = message; // message will be shown on
    // div despite inside another file
```

11.2 Array Extensions

Array Extensions

- Before ES6, when you pass a number to the Array constructor

```
let arr = new Array(3);
console.log(arr);
console.log(arr[0]);
```

- JavaScript creates a array with a length of 3

```
▼(3) [empty × 3] i
    length: 3
  ▶ __proto__: Array(0)
undefined
```

- Note that the array elements will be undefined as only the length of the array is specified

Array Extensions

- However, when you pass a value that's not a number to the array constructor, it instead creates an array with that value inside

```
let arr = new Array("3");
console.log(arr);
console.log(arr[0]);
```

- Sometimes this will cause problems and errors, especially when you might not know the type of data that gets passed into the array constructor

```
▶ ["3"]
3
```

Array Extensions

- The array.of() is a method similar to the array constructor that always creates an array containing the value you passed into it
- No matter what type the element is, it will always create the number of elements equal to the number of arguments passed into the method
- Note that the array.of() method is not a constructor, so don't use the new keyword on it

```
let arr = Array.of(3);
console.log(arr); // [3]
console.log(arr[0]); // 3
```

Array Extensions

- Array.from() creates a new array instance from the passed array

```
let arr = Array.from("ABCDFGH");
console.log(arr);
```

- In case of a string, every character of the string is converted to an element of the new array

```
▶ (7) ["A", "B", "C", "D", "F", "G", "H"]
```

Array Extensions

- Array.from() accepts another argument after the array argument: a callback function that will be executed on every element of the array
- For instance, the example below is an number array ranging from 1 to 10
- What would you do if you want an array with the square root of all the element?

```
let numlist = [1, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10]
```

Array Extensions Activity

- Activity: using array.from() return a new array with all values inside squared
- Create the number array
- Use array from, apply a callback function that multiplies each element by itself (x represents each element in the array)
- Display the result

```
let numlist = [1, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10]
let result = Array.from(numlist, x => x * x);
console.log(result);
```

Array Extensions

- Array.find() is a function that, with a given condition, finds the first valid element in the array

- On the example below is an array containing the age of 10 people
- How would you find the first person that exceed 30 years in age?

```
let agelist = [18, 18, 21, 26, 28, 31, 35, 38, 45, 52];
```

Array Extensions

- Similar to array.from(), array.find() accepts a callback function with a condition that will be applied to every element in an array
- Elements are only valid when they meet the condition stated in the callback function

```
let agelist = [18, 18, 21, 26, 28, 31, 35, 38, 45, 52];
let above30 = agelist.find(x => x > 30);
console.log(above30);
```

```
31
```

Array Extensions

- Array.findIndex() is very similar to array.find()
- It is called on an existing array, and it takes a callback function with conditions as the argument
- The first valid element's index will be returned

```
let agelist = [18, 18, 21, 26, 28, 31, 35, 38, 45, 52];
let above30 = agelist.findIndex(x => x > 30);
console.log(above30);
```

```
5
```

11.3 Object Extensions

Object Extensions

- Object.assign() assigns all properties of an object to a target object
- It takes 2 arguments:
- targetObject to be assigned properties to
- originalObject to have it's properties cloned

```
let name = {
    name: "John"
};

let age = {
    age: 27
};

let person = Object.assign(name, age);
console.log(person);
```

```
▼ {name: "John", age: 27}
    age: 27
    name: "John"
  ▶ __proto__: Object
```

Object Extensions

- Object.assign can be used to clone objects using a empty object on the targetObject argument

- It can also merge objects together

```
let person = {
    name: "John",
    age: 27,
    title: "Salesman"
};

let clonedperson = Object.assign({}, person);
console.log(clonedperson);
```

Object Extensions

- Object.is() is a method that is not restricted to be used with objects
- It works like the === operator with 2 differences:
- -0 and +0
- NaN

- It is not used in most cases as === is simpler to type and the use case is rare

Object Extensions

- The === operator treats -0 and + 0 as the same value
- Object.is() treats them as different values

```
console.log(-0 === +0); // true
console.log(Object.is(-0, +0)); // false

console.log(NaN === NaN); // false
console.log(Object.is(NaN, NaN)); // true
```

- NaN is not equal to NaN using the === operator
- Object.is() treats them as the same

11.4 String Extensions

String Extensions

- ES6 added new methods for string literals, allowing them to be manipulated easier
- The string.startsWith() function checks if a specified string starts with that search string
- The method is called from the target string to be searched
- It takes 2 arguments: the search string to be searched, and the start position to search in index number

```
let hello = "Welcome to JS Tutorial!"
console.log(hello.startsWith("Welcome")); // true
```

String Extensions

- The startsWith method is case-sensitive, so make sure to match exactly as the start of the target string

- The position tells the method where to start, on the example below, the start position is put after the welcome word

```
let hello = "Welcome to JS Tutorial!"
console.log(hello.startsWith("welcome")); // false

console.log(hello.startsWith("to", 8)); // starts from the letter t
```

String Extensions

- String.endsWith() works similarly as startsWith, instead it searches the search string at the end of the target string
- It takes 2 arguments like startsWith: the search string, and the ending position of the target string

```
let hello = "Welcome to JS Tutorial!"
console.log(hello.endsWith("Tutorial!")); // true
```

String Extensions

- String.endsWith is also case-sensitive
- The ending position determines where the target string will end, on the example below, the end position is put before Tutorial
- The position ends before the specified index, the search keyword JS is at index 11 and 12, so the position must be put at 13 to make the searching result true

```
console.log(hello.endsWith("JS", 13)); //ends at the letter S, before the white space
```

String Extensions

- The string.includes() method is another method added in ES6

- It searches the search string inside the target string
- It takes 2 arguments: the searchstring, and the starting position

```
let hello = "Welcome to JS Tutorial!"
console.log(hello.includes("to")); // true
```

String Extensions

- Similar to the other string methods, string.include is case sensitive

- The position determines the start of the searching

```
console.log(hello.endsWith("to", 8)); // starts at the letter t
```

11.5 Recap

ES6 Syntax

- ES6 is a programming language standard for JavaScript, making it more modern and readable
- The let word declares a new variable like var, but it is block-scoped
- A block-scoped variable works well in a asynchronous environment
- Variables declared with the let keyword cannot be referenced outside of its block
- Let doesn't allow you to redeclare a variable
- Let cannot hoist and be referenced before declaration

ES6 Syntax

- The const keyword works similar to the let keyword, except the variable created is read-only
- You cannot reassign a constant, however you can reassign properties inside a constant as it doesn't change the constant itself
- Default Parameters allows for parameters of a function to default to a certain value if no value or undefined is passed on that parameter

ES6 Syntax

- The rest parameter has a prefix of ... and it represents any number of arguments after the rest syntax as an array
- The rest parameter can only be used as the last parameter in a function
- The spread operator has the same prefix ... and it is used to unpack elements inside an array
- The spread operator can be used anywhere inside an array

ES6 Syntax

- If the property name and value of an object is the same, only one has to be used to create an object
- Using square brackets at the property name of an object treats everything inside the brackets as a string, in case of an variable, the value of the variable will be used and treated as string

- The property name of an function inside an object doesn't have to be included in ES6 if the name of the property is the same as the function name

ES6 Syntax

- For…of is a new way of iterating over iterable objects, it works very similar to a for loop, but with much less coding
- Array.entries can be used with for…of to get the index and the value of the element in the array
- Octal Literals have a prefix 0o (zero followed by the letter o) followed by octal numbers
- Binary Literals have a prefix 0b followed by binary numbers
- Template Literals are defined by backticks, it allows for insertion of variables inside a string, as well as allowing single and double quotes inside one

Array Destructuring

- Destructuring assignment allows for assigning multiple variables at the same time using arrays
- The rest syntax can be used on array destructuring, where it takes all the remaining values as an array and assigning it to a variable
- Default parameters can be used to assign a default value to a variable if array destructuring contains a undefined value
- Nested array destructuring is also possible if the variables have the exact same format as the values

Arrow Functions

- Arrow functions has a prefix of an arrow => and it speeds up the creation of functions
- The brackets containing parameters should be used even without parameters before the arrow prefix unless the parameter is the only variable used inside the function
- If the block syntax is used after the arrow prefix, the return keyword will have to be used
- Line breaks are not allowed in arrow functions unless the arrow prefix is placed before the line break
- Modules are extensions of the JS script, that allows you to import and export other methods, variables for use

Assignment

- Follow the assignment paper and finish the assignment in class

References

- Use these if you need more explanations!
- https://www.javascripttutorial.net/es6/
- https://javascript.info/

- Use this if you need more specific answers!
- https://stackoverflow.com/