ES6~ES13新特性（一）

王红元 coderwhy

目录
content

字符串模板基本使用

- 在ES6之前，如果我们想要将字符串和一些动态的变量（标识符）拼接到一起，是非常麻烦和丑陋的（ugly）。
- ES6允许我们使用字符串模板来嵌入JS的变量或者表达式来进行拼接：
  - 首先，我们会使用 `` 符号来编写字符串，称之为模板字符串；
  - 其次，在模板字符串中，我们可以通过 **${expression}** 来嵌入动态的内容；

```
const name = "why"
const age = 18
const height = 1.88

console.log(`my name is ${name}, age is ${age}, height is ${height}`)
console.log(`我是成年人吗? ${age >= 18 ? '是': '否'}`)

function foo() {
  return "function is foo"
}

console.log(`my function is ${foo()}`)
```

标签模板字符串使用

- 模板字符串还有另外一种用法：标签模板字符串（Tagged Template Literals）。
- 我们一起来看一个普通的JavaScript的函数：

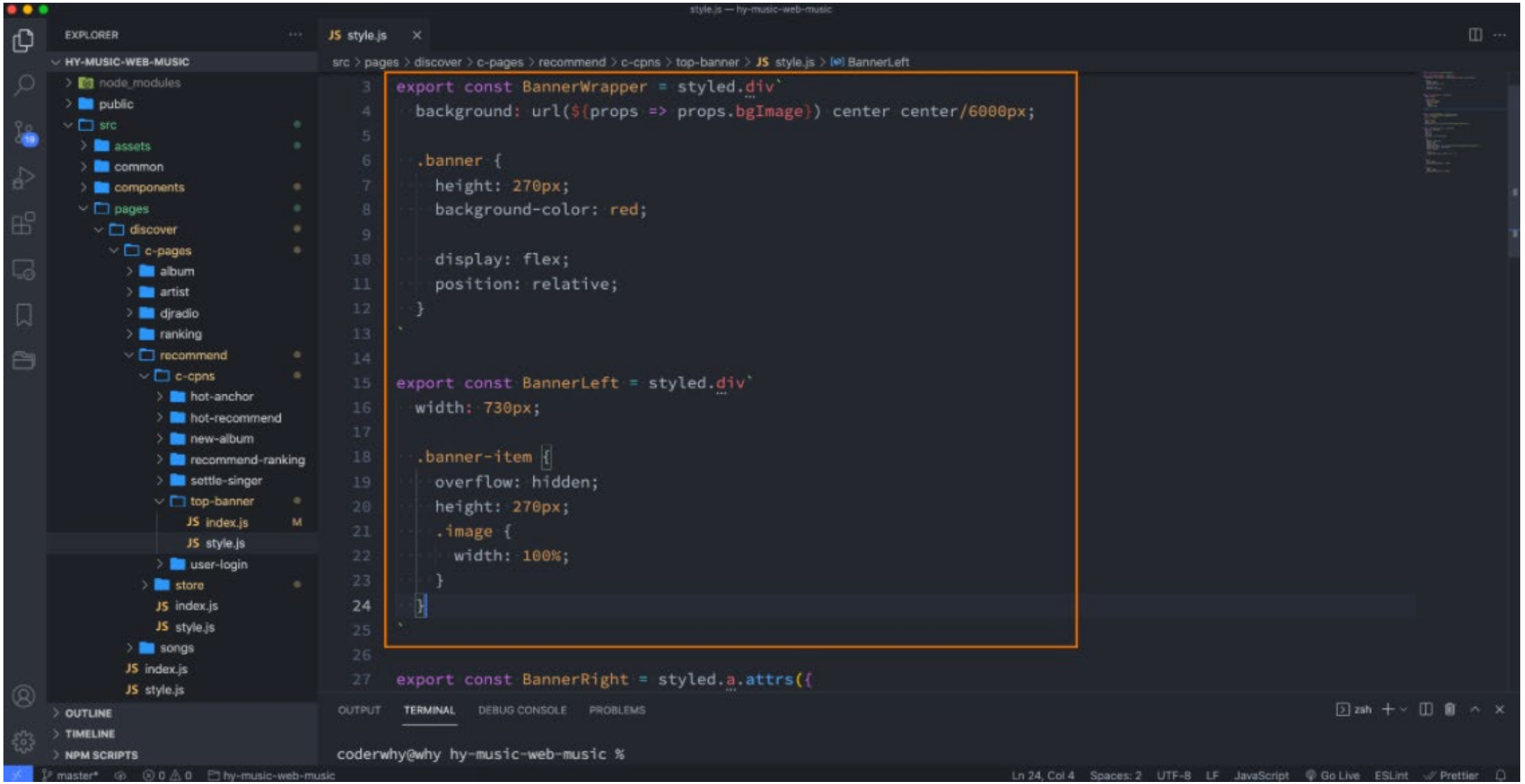
```
function foo(...args) {
  console.log(args)
}

// [ 'Hello World' ]
foo("Hello World")
```

- 如果我们使用标签模板字符串，并且在调用的时候插入其他的变量：
  - 模板字符串被拆分了；
  - 第一个元素是数组，是被模块字符串拆分的字符串组合；
  - 后面的元素是一个个模块字符串传入的内容；

```
const name = "why"
const age = 18
// [ [ 'Hello ', ' World ', '' ], 'why', 18 ]
foo`Hello ${name} World ${age}`
```

React的styled-components库



函数的默认参数

- **在ES6之前，我们编写的函数参数是没有默认值的，所以我们在编写函数时，如果有下面的需求：**
  - 传入了参数，那么使用传入的参数；
  - 没有传入参数，那么使用一个默认值；
- **而在ES6中，我们允许给函数一个默认值：**

```
function foo(x = 20, y = 30) {
  console.log(x, y)
}

foo(50, 100) // 50 100
foo() // 20 30
```

```
function foo() {
  var x =
    arguments.length > 0 && arguments[0] !== undefined ? arguments[0] : 20;
  var y =
    arguments.length > 1 && arguments[1] !== undefined ? arguments[1] : 30;
  console.log(x, y);
}
```

函数默认值的补充

- 默认值也可以和解构一起来使用：

```
// 写法一：
function foo({name, age} = {name: "why", age: 18}) {
  console.log(name, age)
}

// 写法二：
function foo({name = "why", age = 18} = {}) {
  console.log(name, age)
}
```

- 另外参数的默认值我们通常会将其放到最后（在很多语言中，如果不放到最后其实会报错的）：
  - 但是JavaScript允许不将其放到最后，但是意味着还是会按照顺序来匹配；
- 另外默认值会改变函数的length的个数，默认值以及后面的参数都不计算在length之内了。

函数的剩余参数（已经学习）

- **ES6中引用了rest parameter，可以将不定数量的参数放入到一个数组中：**
  - 如果最后一个参数是 ... 为前缀的，那么它会将剩余的参数放到该参数中，并且作为一个数组；

```
function foo(m, n, ...args) {
  console.log(m, n)
  console.log(args)
}
```

- **那么剩余参数和arguments有什么区别呢?**
  - 剩余参数只包含那些没有对应形参的实参，而 arguments 对象包含了传给函数的所有实参；
  - arguments对象不是一个真正的数组，而rest参数是一个真正的数组，可以进行数组的所有操作；
  - arguments是早期的ECMAScript中为了方便去获取所有的参数提供的一个数据结构，而rest参数是ES6中提供并且希望以此来替代arguments的；

- **注意：剩余参数必须放到最后一个位置，否则会报错。**

函数箭头函数的补充

- 在前面我们已经学习了箭头函数的用法，这里进行一些补充：
  - 箭头函数是没有显式原型prototype的，所以不能作为构造函数，使用new来创建对象；
  - 箭头函数也不绑定this、arguments、super参数；

```
var foo = () => {
  console.log("foo")
}

console.log(foo.prototype) // undefined

// TypeError: foo is not a constructor
var f = new foo()
```

14.2.16 Runtime Semantics: Evaluation

ArrowFunction : *ArrowParameters* => *ConciseBody*

1. If the function code for this *ArrowFunction* is strict mode code (10.2.1), let *strict* be **true**. Otherwise let *strict* be **false**.
2. Let *scope* be the LexicalEnvironment of the running execution context.
3. Let *parameters* be CoveredFormalsList of *ArrowParameters*.
4. Let *closure* be FunctionCreate(Arrow, *parameters*, *ConciseBody*, *scope*, *strict*).
5. Return *closure*.

NOTE An *ArrowFunction* does not define local bindings for **arguments**, **super**, **this**, or **new.target**. Any reference to **arguments**, **super**, **this**, or **new.target** within an *ArrowFunction* must resolve to a binding in a lexically enclosing environment. Typically this will

展开语法

- **展开语法(Spread syntax)**：
  - 可以在函数调用/数组构造时，将数组表达式或者string在语法层面展开；
  - 还可以在构造字面量对象时，将对象表达式按key-value的方式展开；
- **展开语法的场景：**
  - 在函数调用时使用；
  - 在数组构造时使用；
  - 在构建对象字面量时，也可以使用展开运算符，这个是在ES2018（ES9）中添加的新特性；
- **注意：展开运算符其实是一种浅拷贝；**

数值的表示

- 在ES6中规范了二进制和八进制的写法：

```
const num1 = 100
// b -> binary
const num2 = 0b100
// octonary
const num3 = 0o100
// hexadecimal
const num4 = 0x100
```

- 另外在ES2021新增特性：数字过长时，可以使用_作为连接符

```
// ES2021新增特性
const num5 = 100_000_000
```

Symbol的基本使用

- Symbol是什么呢？Symbol是ES6中新增的一个基本数据类型，翻译为符号。
- **那么为什么需要Symbol呢?**
  - 在ES6之前，对象的属性名都是字符串形式，那么很容易造成属性名的冲突；
  - 比如原来有一个对象，我们希望在其中添加一个新的属性和值，但是我们在不确定它原来内部有什么内容的情况下，很容易造成冲突，从而覆盖掉它内部的某个属性；
  - 比如我们前面在讲apply、call、bind实现时，我们有给其中添加一个fn属性，那么如果它内部原来已经有了fn属性了呢?
  - 比如开发中我们使用混入，那么混入中出现了同名的属性，必然有一个会被覆盖掉；
- Symbol就是为了解决上面的问题，用来**生成一个独一无二的值。**
  - Symbol值是通过Symbol函数来生成的，生成后可以作为属性名；
  - 也就是在ES6中，对象的属性名可以使用字符串，也可以使用Symbol值；
- **Symbol即使多次创建值，它们也是不同的：**Symbol函数执行后每次创建出来的值都是独一无二的；
- **我们也可以在创建Symbol值的时候传入一个描述description**：这个是ES2019（ES10）新增的特性；

Symbol作为属性名

- 我们通常会使用Symbol在对象中表示唯一的属性名：

```
const s1 = Symbol("abc")
const s2 = Symbol("cba")

const obj = {}

// 1.写法一：属性名赋值
obj[s1] = "abc"
obj[s2] = "cba"

// 2.写法二：Object.defineProperty
Object.defineProperty(obj, s1, {
  enumerable: true,
  configurable: true,
  writable: true,
  value: "abc"
})

// 3.写法三：定义字面量是直接使用
const info = {
  [s1]: "abc",
  [s2]: "cba"
}
```

```
console.log(Object.getOwnPropertySymbols(info))

const symbolKeys = Object.getOwnPropertySymbols(info)
for (const key of symbolKeys) {
  console.log(info[key])
}
```

相同值的Symbol

- **前面我们讲Symbol的目的是为了创建一个独一无二的值，那么如果我们现在就是想创建相同的Symbol应该怎么来做呢?**
  - 我们可以使用Symbol.for方法来做到这一点;
  - 并且我们可以通过Symbol.keyFor方法来获取对应的key;

```
const s1 = Symbol.for("abc")
const s2 = Symbol.for("abc")

console.log(s1 === s2) // true
const key = Symbol.keyFor(s1)
console.log(key) // abc
const s3 = Symbol.for(key)
console.log(s2 === s3) // true
```

Set的基本使用

- 在ES6之前，我们存储数据的结构主要有两种：数组、对象。
  - 在ES6中新增了另外两种数据结构：Set、Map，以及它们的另外形式WeakSet、WeakMap。

- Set是一个新增的数据结构，可以用来保存数据，类似于数组，但是和数组的区别是元素不能重复。
  - 创建Set我们需要通过Set构造函数（暂时没有字面量创建的方式）：
- 我们可以发现Set中存放的元素是不会重复的，那么Set有一个非常常用的功能就是给数组去重。

```
const set1 = new Set()
set1.add(10)
set1.add(14)
set1.add(16)
console.log(set1) // Set(3) { 10, 14, 16 }

const set2 = new Set([11, 15, 18, 11])
console.log(set2) // Set(3) { 11, 15, 18 }
```

```
const arr = [10, 20, 10, 44, 78, 44]
const set = new Set(arr)
const newArray1 = [...set]
const newArray2 = Array.from(set)
console.log(newArray1, newArray2)
```

Set的常见方法

- **Set常见的属性：**
  - size：返回Set中元素的个数；
- **Set常用的方法：**
  - add(value)：添加某个元素，返回Set对象本身；
  - delete(value)：从set中删除和这个值相等的元素，返回boolean类型；
  - has(value)：判断set中是否存在某个元素，返回boolean类型；
  - clear()：清空set中所有的元素，没有返回值；
  - forEach(callback, [, thisArg])：通过forEach遍历set；
- **另外Set是支持for of的遍历的。**

WeakSet使用

- **和Set类似的另外一个数据结构称之为WeakSet，也是内部元素不能重复的数据结构。**
- **那么和Set有什么区别呢?**
  - 区别一：WeakSet中只能存放对象类型，不能存放基本数据类型；
  - 区别二：WeakSet对元素的引用是弱引用，如果没有其他引用对某个对象进行引用，那么GC可以对该对象进行回收；

```
const wset = new WeakSet()

// TypeError: Invalid value used in weak set
wset.add(10)
```

- **WeakSet常见的方法：**
  - add(value)：添加某个元素，返回WeakSet对象本身；
  - delete(value)：从WeakSet中删除和这个值相等的元素，返回boolean类型；
  - has(value)：判断WeakSet中是否存在某个元素，返回boolean类型；

WeakSet的应用

- **注意：WeakSet不能遍历**
  - 因为WeakSet只是对对象的弱引用，如果我们遍历获取到其中的元素，那么有可能造成对象不能正常的销毁。
  - 所以存储到WeakSet中的对象是没办法获取的；
- **那么这个东西有什么用呢?**
  - 事实上这个问题并不好回答，我们来使用一个Stack Overflow上的答案；

```
const pwset = new WeakSet()
class Person {
  constructor() {
    pwset.add(this)
  }
  running() {
    if(!pwset.has(this)) throw new Error("不能通过其他对象调用running方法")
    console.log("running", this)
  }
}
```

Map的基本使用

- 另外一个新增的数据结构是Map，用于**存储映射关系**。
- 但是我们可能会想，在之前我们可以**使用对象来存储映射关系，他们有什么区别**呢?
  - 事实上我们对象存储映射关系只能用字符串（ES6新增了Symbol）作为属性名（key）；
  - 某些情况下我们可能希望通过其他类型作为key，比如对象，这个时候会自动将对象转成字符串来作为key;
- 那么我们就可以使用Map:

```
const obj1 = { name: "why" }
const obj2 = { age: 18 }

const map = new Map()
map.set(obj1, "abc")
map.set(obj2, "cba")
console.log(map.get(obj1))
console.log(map.get(obj2))
```

```
const map = new Map([
  [obj1, "abc"],
  [obj2, "cba"],
  [obj1, "nba"]
])
console.log(map.get(obj1)) // nba
console.log(map.get(obj2)) // cba
```

Map的常用方法

- **Map常见的属性:**
  - size：返回Map中元素的个数;
- **Map常见的方法:**
  - set(key, value)：在Map中添加key、value，并且返回整个Map对象;
  - get(key)：根据key获取Map中的value;
  - has(key)：判断是否包括某一个key，返回Boolean类型;
  - delete(key)：根据key删除一个键值对，返回Boolean类型;
  - clear()：清空所有的元素;
  - forEach(callback, [, thisArg])：通过forEach遍历Map;
- **Map也可以通过for of进行遍历。**

WeakMap的使用

- **和Map类型的另外一个数据结构称之为WeakMap，也是以键值对的形式存在的。**
- 那么和Map有什么区别呢?
  - 区别一：WeakMap的key只能使用对象，不接受其他的类型作为key；
  - 区别二：WeakMap的key对对象想的引用是弱引用，如果没有其他引用引用这个对象，那么GC可以回收该对象；

```
const weakMap = new WeakMap()
// Invalid value used as weak map key
weakMap.set(1, "abc")
// Invalid value used as weak map key
weakMap.set("aaa", "cba")
```

- **WeakMap常见的方法有四个：**
  - set(key, value)：在Map中添加key、value，并且返回整个Map对象；
  - get(key)：根据key获取Map中的value；
  - has(key)：判断是否包括某一个key，返回Boolean类型；
  - delete(key)：根据key删除一个键值对，返回Boolean类型；

WeakMap的应用

- **注意：WeakMap也是不能遍历的**
  - 没有forEach方法，也不支持通过for of的方式进行遍历；
- **那么我们的WeakMap有什么作用呢？（后续专门讲解）**

```
// WeakMap({key(对象): value}): key是一个对象，弱引用
const targetMap = new WeakMap();
function getDep(target, key) {
  // 1.根据对象(target)取出对应的Map对象
  let depsMap = targetMap.get(target);
  if (!depsMap) {
    depsMap = new Map();
    targetMap.set(target, depsMap);
  }

  // 2.取出具体的dep对象
  let dep = depsMap.get(key);
  if (!dep) {
    dep = new Dep();
    depsMap.set(key, dep);
  }
  return dep;
}
```

ES6其他知识点说明

- **事实上ES6（ES2015）是一次非常大的版本更新，所以里面重要的特性非常多：**
  - 除了前面讲到的特性外还有很多其他特性；
- **Proxy、Reflect，我们会在后续专门进行学习。**
  - 并且会利用Proxy、Reflect来讲解Vue3的响应式原理；
- **Promise，用于处理异步的解决方案**
  - 后续会详细学习；
  - 并且会学习如何手写Promise；
- **ES Module模块化开发：**
  - 从ES6开发，JavaScript可以进行原生的模块化开发；
  - 这部分内容会在工程化部分学习；
  - 包括其他模块化方案：CommonJS、AMD、CMD等方案；