ES6简介

ES6， 全称 ECMAScript 6.0 ，是 JavaScript 的下一个版本标准，2015.06 发版。

ES6 主要是为了解决 ES5 的先天不足，比如 JavaScript 里并没有类的概念，但是目前浏览器的 JavaScript 是 ES5 版本，大多数高版本的浏览器也支持 ES6，不过只实现了 ES6 的部分特性和功能。

例如箭头函数(arrow functions)和简单的字符串插值(string interpolation),大到烧脑的新概念,例如代理(proxy)和生成器(generators) 等，经常使用，才更熟悉。

let与const和块级作用域

es5的特点

- var 与 function 存在变量提升
- var 只会提前声明，function 既声明又定义
- 在全局作用域下，使用 var 和 function 声明的变量会给 window 增加属性

es6的特点

let

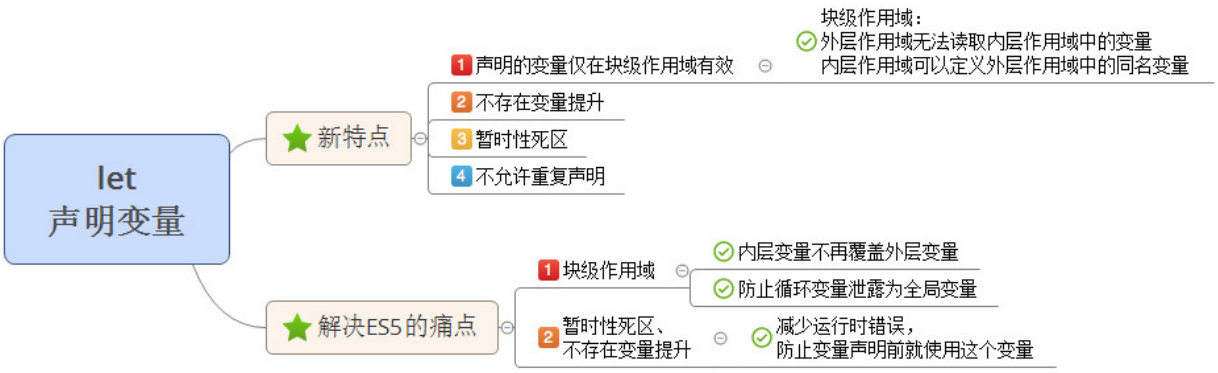
- 使用 let 没有变量提升
- 不可以重复声明：即在`let a = 1`之后，不可以再`let a = 2`
- 不会给 window 增加属性

const

- 没有变量提升
- 不可以重复声明
- 不会给 window 增加属性
- const 定义变量，一旦声明必须赋值
- const 定义的是一个常量，不可以重新赋值

解决的痛点

1. 块级作用域
2. 不存在变量提升（减少运行时错误，防止变量声明前就使用这个变量）



字符串的扩展

常用方法

1. includes
2. startWith
3. endWith
4. repeat

```
var str = "222";
str.repeat(2); // 222222
```

模版字符串：${}

数组的扩展

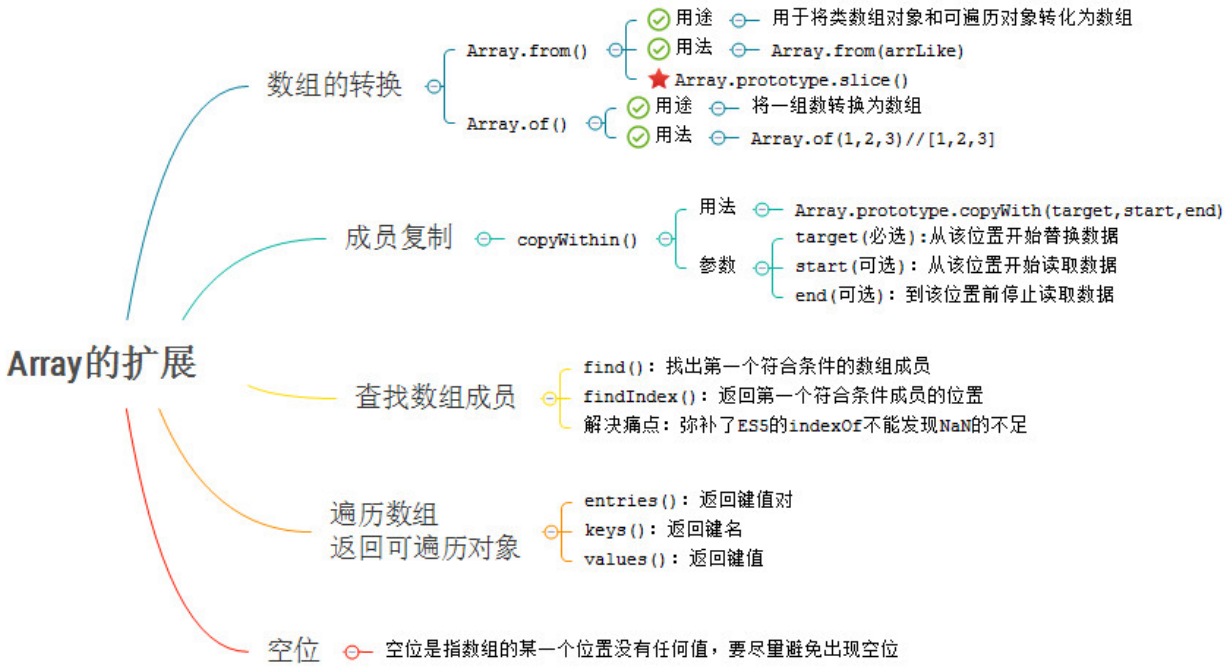
1. 数组的转换：Array.from(),Array.of() 去重，转换累数组
2. 查找数组：find(),findIndex(),includes(),(解决es5的indexOf不能发现NaN的不足)
3. 遍历：entries,keys,values

```
var arr = ['a', 'b', 'c'];
var iterator = arr.keys();

console.log(iterator.next()); // { value: 0, done: false }
console.log(iterator.next()); // { value: 1, done: false }
console.log(iterator.next()); // { value: 2, done: false }
console.log(iterator.next()); // { value: undefined, done: true }
```

4. 数组空位：指数组的某一位置好没有任何值

解决的痛点：

includes(),(解决es5的**indexOf**不能发现**NaN**的不足



解构赋值

1. 数组的解构赋值

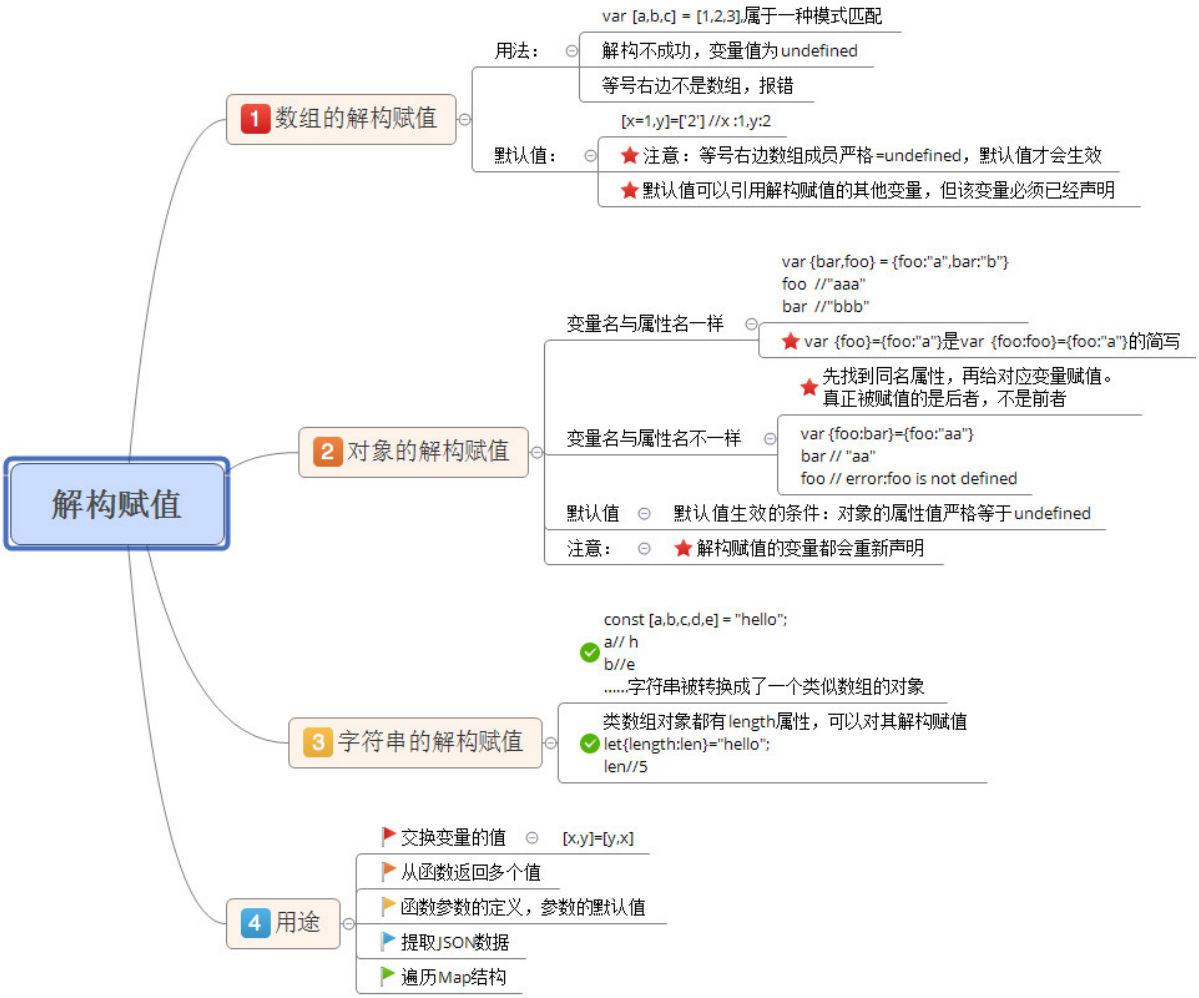
```
let arr = [1,2,3,4];
let [a,b,c,d] = arr;
console.log(a,b,c,d);// 打印出每个值
```

2. 对象的解构赋值

```
let {name:name,age:age}={name:"Cyan",age:19};
// 等价于
let {name,age} = {name:"Aqing",age:20};
```

3. 扩展运算符

```
...

[...arr1,...arr2]
```



函数的扩展

1. 参数解构赋值。

```
function fn(x="A",y="C") {
  console.log(x + y);
}
```

2. 参数作用域

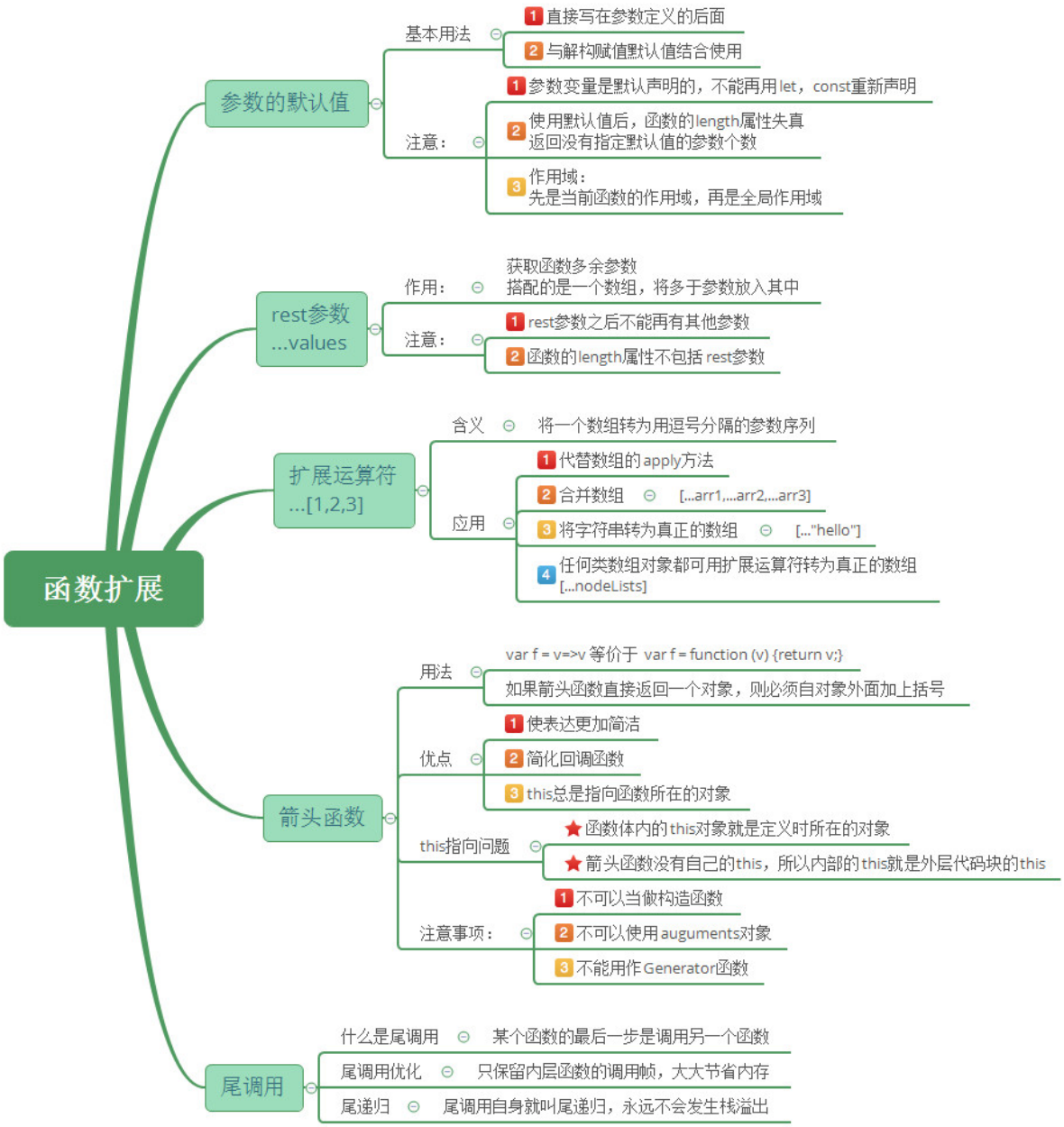
```
let a = 1, b = 2;
function fn1(x = m,y = n) {
  // 私有作用域：私有变量 x,y
  // 进入函数时先给形参赋值，发现无私有变量m跟n，遂找到全局的
  console.log(x);
  console.log(y);
  let m = "kkkk"; // 改用var 结果与let一样
  let n = "ES6";
}
fn1(); // 1 2
fn1(100); // 100 2 说明默认值用的是全局的
```

3. reset
4. 箭头函数

```
let fn = (x, y) => {}

// 箭头函数this指向
let obj = {
  fn:function () {let f = () => {console.log(this);};
  f();
 }
};
obj.fn(); // {fn: ƒ}

// 箭头函数没有arguments
let f1 = (...arg)=>{
  // console.log(arguments);
  console.log(arg);
};
// f1(1,23);//报错：arguments is not defined
f1(1,2,3); // [1, 2, 3]
```

注意的问题：

- 若函数体只有一行代码的话，就可以省略`{}`，若只有一个参数，就可以省略小括号。
- 通常函数当做参数的时候(回调函数)使用箭头函数
- 箭头函数没有this指向，它里面的this是上一级的作用域
- 箭头函数没有arguments
- 箭头函数不可以用作构造函数 因为不可以使用new执行



对象的扩展

1. 简洁写法

```
let name = 'liuyi', age = 20;
let obj = {name, age};
console.log(obj); // {name: "liuyi", age: 20}
```

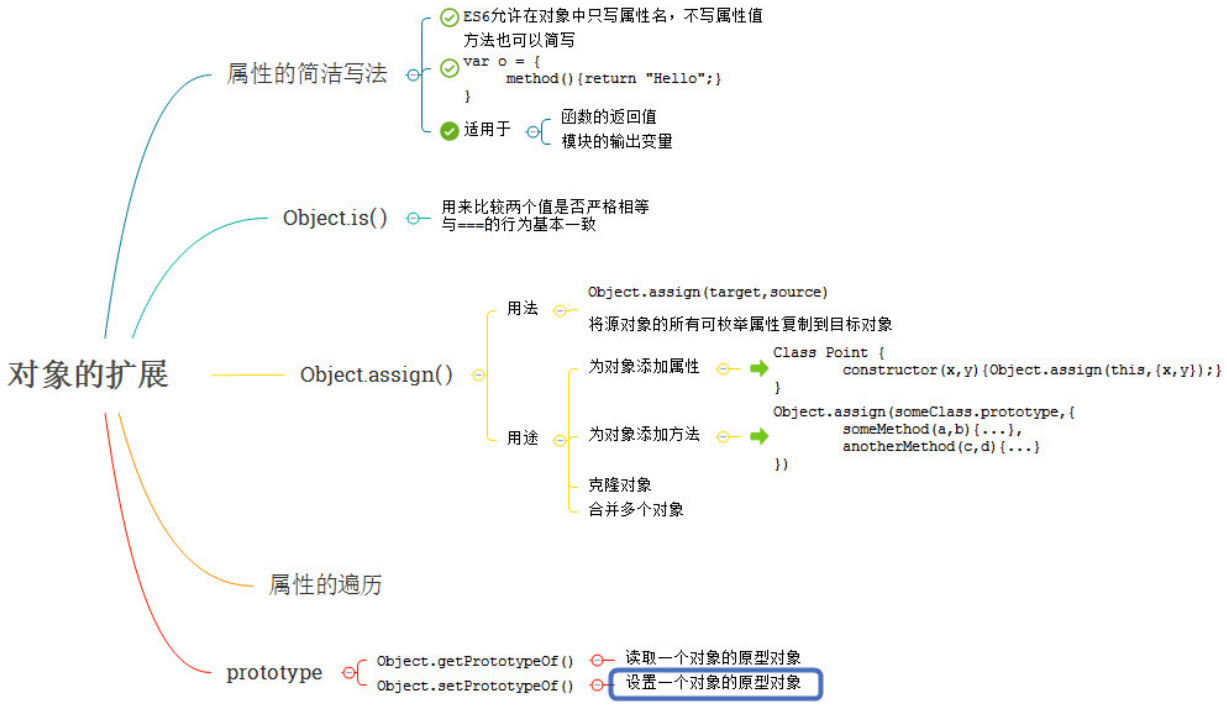
2. Object.is(),与===的区别
3. Object.assign()。。。获取当前行的数据后（row的数据如果改变了，会影响表格的数据源）

4. 遍历

```
console.log(Object.keys(obj)); // ["name", "age"]

// 经常看到一些代码
Object.keys(obj).forEach(item => {})


console.log(Object.values(obj)); // ["liuyi", 20]



console.log(Object.entries(obj)); // [Array(2), Array(2)]  展开后0:(2)
["name", "liuyi"]   1:(2) ["age", 20]
```

5. 对象的get与set函数. 对象属性的设置与获取都会触发它内置的set与get函数，我们也可以显式控制这两个函数

```
let obj = {
  _name:"liuyi",
  get name(){
    // 只要通过obj获取name属性就会触发这个函数
    // 可以通过return 返回值
    console.log(1);
    return this._name;
  },
  set name(val){
    // 只要通过obj给name属性设置值就会触发set函数
    console.log(2);
    // val：设置的值
    this._name = val;
  }
};

// Vue中的计算属性，直接赋值会warning
```



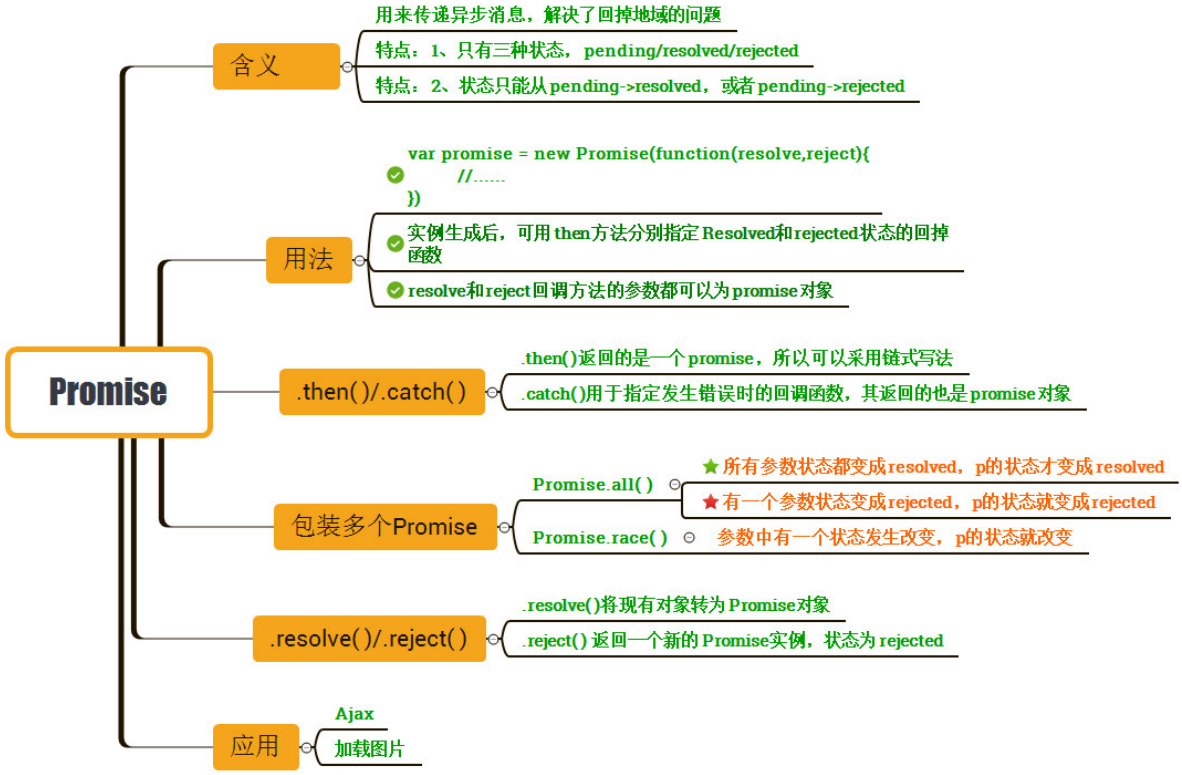
Promise

- Promise的实例分为三个状态，一开始的状态就是pending（等待）状态，一旦new后，立马执行函数。
- 执行函数的顺序：new Promise中的代码 ===> 当前队列中的同步代码 ===> then(异步)里面的回调函数
- Promise的实例另外两个状态是：reslove（成功）、reject（失败），他们在代码中是具体的两个用作回调的函数。
- 实例使用`.then`来调用reslove或者reject函数，若成功，then方法里执行的函数就是resolve，失败执行的就是reject。

```
let pro1 = new Promise((resolve,reject)=>{
  //如果在new Promise中有错误，那么会直接执行then中的第二个回调函数，并且把错误信息传给函数
  resolve("success");
  reject("error");
});

//then方法有两个回调函数
pro1.then((res)=>{
  console.log(res);
  }, (e)=>{
  //失败的回调
  console.log(e);
}); // 成功打印出了success，如果我们把resolve函数注释掉，那么就会打印e
```

```
console.log("因为then方法是异步的，所以不会等待，跳过直接进行这里的代码,所以这里先执行");
```

举例：CRM项目中，axios，全局请求拦截

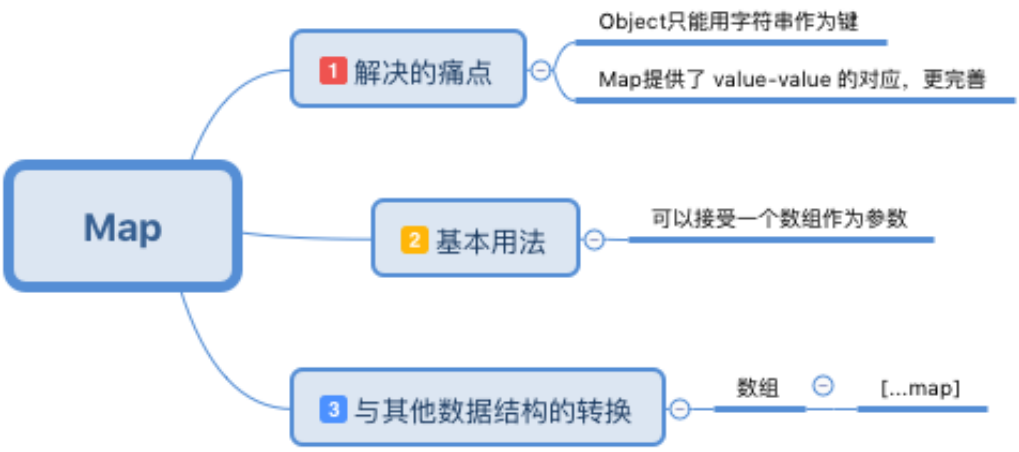


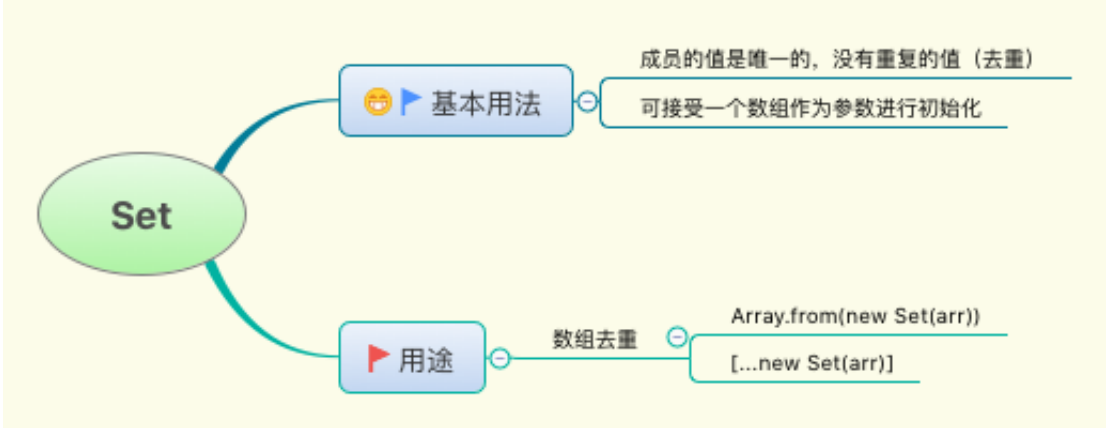
Symbol，Set，Map

1. Set去重

```
Array.from(new Set(arr))

[...new Set(arr)]
```

2. map.提供了“值-值”对应，更完善





Class

定义类

类实际上是个“特殊的函数”，就像你能够定义的函数声明和函数表达式一样，类语法有两个组成部分：类表达式和类声明

类声明

```
class Car {
  constructor(color, size) {
    this.color = color;
    this.size = size;
  }
}
```

类表达式

```
/* 匿名类 */
let Car = class {
  constructor(height, width) {
    this.height = height;
    this.width = width;
```

```
  }
};

/* 命名的类 */
let Car = class Rectangle {
  constructor(color, size) {
    this.color = color;
    this.size = size;
  }
};
```

构造函数

`constructor`方法是一个特殊的方法，这种方法用于创建和初始化一个由`class`创建的对象。一个类只能拥有一个名为 “constructor”的特殊方法。

能用`super` 关键字来调用一个父类的构造函数

原型方法

```
class Count {
    // constructor
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    // Getter
    get sum() {
        return this.mul()
    }
    // Method
    mul() {
        return this.x * this.y;
    }
}
const mul = new Rectangle(10, 10);

console.log(mul.sum);
// 100
```

静态方法

```
class Point {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
```

```
    static distance(a, b) {
        const dx = a.x - b.x;
        const dy = a.y - b.y;

        return Math.hypot(dx, dy);
    }
}

const p1 = new Point(5, 5);
const p2 = new Point(10, 10);

console.log(Point.distance(p1, p2));
```

总结

目前常用的有

1. let const
2. Array.includes()
3. Object.assign()
4. 扩展运算符。[...arr1,...arr2]合并数组
5. Array.from(new Set(arr))，[...new Set(arr)] 数组去重
6. Object.keys(),Object.values()

还有Proxy，Iterator，Generator等，未完，下一期