# 面试题

### 基本数据类型

String、Number、undefined、null、Boolean

1. **undefined和null区别；notdefined和undefined区别**

* 1. null:表示没有对象，即该处不该有值；
     1. 作为函数的参数，表示函数的参数不是对象
     2. 作为原型链对象的终点
  2. undefined表示缺少值，此时应该有值但是还没有被赋值
     1. 变量被声明了，没有赋值
     2. 调用函数应提供的参数没有提供
     3. 对象没有赋值的属性
     4. 函数没有返回值时默认返回undefined

* 1. Undefind是js的一种数据类型；not defined遇到没有定义的变量但是使用了，是一种错误

1. **数组常用的方法**

forEach/push/pop/unshift/shift/concat/filter/map/every/indexof/join/reverse/sort/slice/splice/reduce

1. **数组map和forEach的区别：**

Map返回新的数组，不会改变原数组数组中的元素为原始数组调用处理函数后的值；且不会对空数组进行检测

forEach改变的是原数组，对于空数组不会调用回调函数

1. **对象的属性**
   1. Object.getPrototypeOf(obj)返回指定对象的原型,参数时要返回原型的对象：es5
   2. Object.setPrototypeOf(现有对象, 原型对象),为现有对象设置原型，返回一个新对象
   3. Object.create(obj)创建一个新对象，使用现有的对象来提供新创建的对象的\_\_proto\_\_,参数是新创建对象的原型对象；es5
2. **ES6新特性**

String新增的方法（重点方法）

* 1. Includes(val)：返回布尔值，是否找到了参数字符串
  2. startsWith(val)：返回布尔值，是否以某字符开头
  3. endsWith(val)：返回布尔值，是否以某字符结尾
  4. repeat(val)：表示将某个字符重复几遍，返回新字符串；’x’.repeat(3) => ‘xxx’
     1. 参数是小数，会被向下取整
     2. 参数时负数或者infinity，会报错
     3. 0到-1之间的小数等同于0
     4. 参数NaN等同于0
     5. 参数字符串，会先转成数字
  5. trimStart/trimEnd：消除头部和尾部的空格，返回新字符串，不会修改原字符串
  6. matchAll（）：返回一个正则表达式在当前字符串的所有匹配
  7. padStart/padEnd
     1. padStart:用于头部补全，padEnd: 用于尾部补全
     2. 参数1：字符串补全生效的最大长度；参数2：用于补全的字符串
     3. ‘x’.padStart(5, ‘ab’) => ababx; ‘x’.padStart(4, ‘ab’) => abax;
     4. 如果原字符串的长度大于有效的最大长度，用于补全的字符串不生效。原字符串和补全字符串之和大于长度，截取超出位数，省略补全字符串，则用空格代替
     5. 用途提示字符串格式或为数值补全指定位数

**数值相关（重点）**

1. Number.isFinite()用来检查一个数值是否为有限的（finite） / Numer.isNaN()
   1. 区别于原先的sFinite()、isNaN()只对数值有效
2. Number.parseInt()/Number.parseFloat()
   1. 行为与原先保持不变
   2. 用法要用Number点出来
3. Number.isInterger()：判断数值是否为整数；不建议使用Number.isInteger()
   1. 参数不是数值返回false
   2. Number.isInterger(5.0)：true；原因：整数和浮点数采用的是同样的储存方法
   3. 参数失去精度和小于JavaScript能识别的最小值，可能会误判；
4. Math.trunc(val):用于去除数值的小数部分，返回整数部分
   1. 非数值，内部先转化为数值
   2. 空值或者无法截取的值，返回NaN
5. Math.sign(val): 方法用来判断一个数到底是正数、负数、还是零。对于非数值，会先将其转换为数值。
   1. 参数为正数，返回+1；
   2. 参数为负数，返回-1；
   3. 参数为 0，返回0；
   4. 参数为-0，返回-0;
   5. 其他值，返回NaN
6. BigInt数据类型：大整数，只表示整数，没有位数限制，
   1. BigInt 类型的数据必须添加后缀n
   2. 只能使用负号，不能使用正号

7） BigInt对象

**函数相关（重点）**

1. 允许为函数参数设置默认值
2. reset参数，形式为(…变量名),用于获取函数多余的参数，代表的是真正的数组
   1. reset参数后不能再有其他参数
3. 与解构赋值默认值使用
4. 严格模式
   1. 规定只要函数参数使用了默认值、解构赋值、或者扩展运算符，那么函数内部就不能显式设定为严格模式，否则会报错
5. 箭头函数
   1. 函数体内的this， 就是定义所在的对象，而不是使用时的对象
      1. This对象的指向是可以可变的， 在箭头函数中this是固定的，所以箭头函数根本没有自己的this，内部代码块的this就是外层代码块的this
   2. 不可以当做构造函数，即不能new
   3. 不能使用arguments对象，但是可以用reset参数
   4. 不可以使用yield命令，因此箭头函数不能用作 Generator 函数。
   5. 定义对象的方法时不能使用箭头函数，需要动态this的时候也不能使用
6. 函数尾调：函数的最后一步是调用另一个函数
   1. 如果尾调用自身，就称为尾递归
   2. ES6 的尾调用优化只在严格模式下开启，正常模式是无效的

**数组相关（重点）**

1. 任何定义了遍历器（Iterator）接口的对象（参阅 Iterator 一章），都可以用扩展运算符转为真正的数组。
2. Array.from() 将伪数组或者类似数组的对象（只要有length属性）转化为真正的数组
3. Array.of(), 将一组数值转为数组 Array.of(1, 2, 3) => [1, 2, 3]
4. 实例方法
   1. copyWithin(target[必填], start, end):在当前数组内部，将指定的成员复制到其他位置（会覆盖所有成员），然后返回当前数组，即会修改当前数组
      1. target：从target位置开始替换数据，负数就是倒着来
      2. start：读取数据开始的位置
      3. end：到该位置前结束
   2. find() /findIndex()
      1. find参数为回调函数，所有的成员执行这个回调函数，直到找到返回true的结果，没有找到则返回undefined
      2. findIndex（）与find相似，找到的是第一个符合条件的位置，没有符合条件的返回-1
   3. fill() 使用给定值，填充数组
      1. 参数1：需要填充的值
      2. 参数2（可选）：从什么位置开始
      3. 参数3（可选）：从什么位置结束
   4. Includes()：返回一个布尔值，数组是否包含某个值
      1. 参数1：是否存在的值
      2. 参数2（可选）：搜索的开始位置，默认0，负数表示倒数的位置
   5. entries()/keys()/values();用于遍历数组
   6. flat()/flatMap()
      1. flat用于将嵌套的数组拉平，变为一维数组；[1, [2], 3].flat() => [1, 2, 3],flat只会拉平一层，如果是多层嵌套，可以将flat的参数写为整数，默认1
      2. flatMap(回调函数)，每个成员都执行回调函数，然后对返回值进行flat方法，返回新数组，不改变原数组，还可以有第二个参数，用来绑定遍历函数的this

4. 数组的空位: 数组的某一个位置没有任何值；entries()、keys()、values()、find()和findIndex()会把空位处理成undefined

5. 数组去重的方法

* 1. 去除重复成员方法 […new Set(array)]
  2. 数组去重new Set([…])结合Array.from（）:即Array.from（）将去重后的set集合，转化为数组类型
  3. 数组的扩展运算符和Set结构结合 let arr = [x, b, c…] let unique = […new Set(arr)]

**对象相关（重点）**

1）属性的简洁表示：const foo = ‘bar; const obj = {foo}

2）属性名表达式：es6允许将属性名写成表达式，需要用[]包起来

Const obj = {[‘name’ + ‘1’] : ‘jack’}

属性名表达式不能和简洁方式一起使用

属性名表达式如果是一个对象，默认会转成字符串[Object, Object]

1. 方法的name属性：var obj = {test() {}}, obj.test.name = ‘test’
   1. bind方法创造的函数，name属性返回bound加上原函数的名字；
   2. Function构造函数创造的函数，name属性返回anonymous。
2. Super关键字指向当前对象的原型对象
   1. super关键字表示原型对象时，只能用在对象的方法之中，用在其他地方都会报错。
3. 扩展预算符…
4. 新增的方法：
   1. Object.is()它用来比较两个值是否严格相等;不同之处+0不等于-0；NaN等于自身
      1. 用法Object(+0, -0) false
   2. Object.assign()用于对象的合并，将源对象的所有可枚举属性，复制到目标对象
      1. 是浅拷贝
      2. 第一个参数时目标对象。其余参数都是源对象。若源对象和目标对象、多个源对象的属性重名，后边的值会覆盖前面的值
      3. 只有一个参数会直接返回该参数
      4. 参数不是对象会转成对象，然后返回
      5. Undefined和null无法转成对象，会报错
      6. 可以处理数组，但是会把数组看成对象
      7. 只能进行值的复制，如果复制一个取值函数，会先取值，再复制
      8. 常见用途
         1. 为对象添加属性
         2. 为对象添加方法
         3. 克隆对象（不能克隆继承的值）
         4. 合并多个对象
         5. 为属性设置默认值
   3. Object.getOwnPropertyDescriptors()，返回指定对象所有自身属性（非继承属性）的描述对象
   4. Object.keys()，Object.values()，Object.entries()
      1. Object.keys()返回一个数组，成员是参数对象自身的（不含继承的）所有可遍历（enumerable）属性的键名。
      2. Object.values()成员是参数对象自身的（不含继承的）所有可遍历（enumerable）属性的键值。
      3. 方法返回一个数组，成员是参数对象自身的（不含继承的）所有可遍历（enumerable）属性的键值对数组。Object.entries(obj) =>[ ["foo", "bar"], ["baz", 42] ]
   5. Object.fromEntries();是Object.entries()的逆操作，用于将一个键值对数组转为对象。
      1. 特别适合将 Map 结构转为对象。

6) 属性的可枚举性和遍历

a) 可枚举性不会！

b) 方法

1. For…in
2. Object.keys(obj); 返回一个数组，包含对象（不包括继承）的所有可枚举性的键名；可用来判断对象是否为空
3. Object.getOwnPropertyNames(obj)：返回一个数组，包含对象自身的所有属性（不含 Symbol 属性，但是包括不可枚举属性）的键名。
4. Object.getOwnPropertySymbols(obj)返回一个数组，包含对象自身的所有 Symbol 属性的键名。
5. Reflect.ownKeys(obj)：返回一个数组，包含对象自身的（不含继承的）所有键名，不管键名是 Symbol 或字符串，也不管是否可枚举

**Symbol（重点）**

1. Symbol是es6添加的新的原始数据类型，表示独一无二的值
2. Symbol值通过Symbol对象生成
3. Symbol函数前不能使用new命令，否则报错；这是因为生成的 Symbol 是一个原始类型的值，不是对象，是一种类似于字符串的数据类型。
4. 可以接受一个参数，作为Symbol实例的描述，主要是为了控制台显示或者转字符串时比较好区分
   1. 例如：const a = Symbol(‘test’);console.log(a) => ‘Symbol(test)’
   2. 如果参数是对象，就会调用该对象的toString方法，转为字符转，然后生成Symbol值
   3. Symbol的值不能与其他类型的数值进行运算，否则会报错，但是可以显式转为数值，ley a = Symbol(‘test’);String(a);/a.toString().也可以转布尔值，但是不能转数值；Boolean（a）
5. Symbol.prototype.description; 实例属性description用来，获取Symbol的描述
   1. Let a = Symbol(‘test’); a.description; => ‘test’
6. 作为对象属性名的Symbol；不能使用点运算符，要obj[Symbol的值]
7. 还可以定义一组常量，保证这组常量的值都是不相等的
8. 消除魔术字符串：魔术字符串是指代码中多次出现，与之形成强耦合的某一个具体的字符串或者数值
9. 属性名的遍历
   1. Symbol作为属性名，遍历的时候不会出现for…in等方法中，但是也不是私有属性，Object.getOwnPropertySymbols()，返回一个数组，成员是当前对象的所有用作属性名的Symbol值
10. Symbol.for()，Symbol.keyFor()

**Set和Map结构（重点）**

1. Set数据结构：类似与数组，但是成员的值都是唯一的，没有重复的值
2. Set本身是一个构造函数，用来生成Set数据结构： const s = new Set()
3. Set函授可以接受一个数组作为参数，用于初始化；const s = new Set([……])
4. 向set添加值的时候，不会发生类型转换；另外两个对象总是不相等的
5. Set实例的属性和方法
   1. 实例的属性
      1. Set.prototype.constructor: 构造函数，默认是Set函数
      2. Set.pototype.size:返回set实例的成员总数
   2. 方法分为操作方法和遍历方法
      1. 操作方法：
         1. Set.prototype.add(value);添加某个值，返回Set结构本身
         2. Set.prototype.delete(value);删除一个值，返回布尔值，是否删除成功
         3. Set.prototype.has(value);返回布尔值，表示该值是否为set的成员
         4. Set.prototype.clear();清除所有成员，无返回值
      2. 遍历方法（遍历顺序是插入顺序）
         1. Set.prototype.keys()返回键名的遍历器
         2. Set.prototype.values()返回键值的遍历器
         3. Set.prototype.entries()返回键值对的遍历器
         4. Set.prototype.forEach()使用回调函数遍历每个成员
            1. forEach方法出第一个回调函数参数除外，可有第二个参数，表示绑定处理函数内部的this对象
      3. 由于 Set 结构没有键名，，所以keys方法和values方法的行为完全一致。
      4. Set结构的数据默认可遍历，直接用for…of循环遍历
      5. 也可以使用扩展运算符，使用扩展运算符后可以与map和filter方法
6. WebSet（不可遍历）：结构与set相似
   1. 区别
      1. 成员只能是对象
      2. 成员都是弱引用；即垃圾回收机制不考虑 WeakSet 对该对象的引用，也就是说，如果其他对象都不再引用该对象，那么垃圾回收机制会自动回收该对象所占用的内存，不考虑该对象还存在于 WeakSet 之中
         * 1. WeakSet 适合临时存放一组对象，以及存放跟对象绑定的信息。只要这些对象在外部消失，它在 WeakSet 里面的引用就会自动消失。
   2. 是一个构造函数，可以使用new命令；可以接受一个数组或者类似数组的对象作为参数，数组的成员只能为对象，因为是数组的成员成为webSet的成员，不是数组本身
   3. 方法
      1. WeakSet.prototype.add(value)：向 WeakSet 实例添加一个新成员。
      2. WeakSet.prototype.delete(value)：清除 WeakSet 实例的指定成员。
      3. WeakSet.prototype.has(value)：返回一个布尔值，表示某个值是否在 WeakSet 实例之中。
7. Map，类似对象，也是键值对的集合，但是键不只限于字符串，也可以接受数组作为参数，该数组的成员是表示一个个表示键值对的数组，任何具有 Iterator 接口、且每个成员都是一个双元素的数组的数据结构（详见《Iterator》一章）都可以当作Map构造函数的参数。
   1. 如果对一个键多次赋值，后面的会覆盖前面的值
   2. 读取未知的键，结果是undefined
   3. 只有对同一个对象的引用，Map 结构才将其视为同一个键;由此可知，Map的键实际上跟内存地址绑定的。
      1. Const Map = new Map(); map.set([‘a’]); map.get([‘a’]) => undefined
      2. 则只要两个值严格相等，Map将视为一个键
   4. 实例的属性和操作方法
      1. Size属性，返回Map结构的成员总数；const map = new Map(); map.size
      2. Map.prototype.set(key, value);设置key对应的键值为value，然后返回整个map结构，如果key已经有值，键值会被更新
         1. Set方法返回的是当前Map对象，因此可以链式写法
      3. Map.prototype.get(key) get方法读取key对应的键值，如果找不到key，返回undefined
      4. Map.prototype.has(key) has方法返回一个布尔值，表示某个键是否在当前 Map 对象之中
      5. Map.prototype.delete(key) delete方法删除某个键，返回布尔值。
      6. Map.prototype.clear()clear方法清除所有成员，没有返回值。
   5. 遍历方法（Map 的遍历顺序就是插入顺序）
      1. Map.prototype.keys()：返回键名的遍历器。
      2. Map.prototype.values()：返回键值的遍历器。
      3. Map.prototype.entries()：返回所有成员的遍历器。
      4. Map.prototype.forEach()：遍历 Map 的所有成员。
   6. 转数组结构
      1. 扩展运算符

d) 数组转map，将数组传入map构造函数

e) Map转对象

f) 对象转为 Map 可以通过Object.entries()

9. WeakMap

* 1. 只接受对象作为键名，不接受其他类型的值作为键名
  2. WeakMap的键名所指向的对象，不计入垃圾回收机制
  3. WeakMap 就是为了解决内存泄漏而诞生的，它的键名所引用的对象都是弱引用，即垃圾回收机制不将该引用考虑在内。因此，只要所引用的对象的其他引用都被清除，垃圾回收机制就会释放该对象所占用的内存

**class（重点）**

1. Class类必须使用new调用，实例的属性必须显示的定义在其本身上，否则都是定义在原型上
2. 取值getter和setter函数，在类的内部可以使用get和set关键字，对于某个属性设置存取时的行为进行拦截
3. 注意点
   1. 类和模块内部，默认是严格模式
   2. 不存在变量的提升
   3. 有name属性，.name获取类的名称
   4. 某个方法前加上\*，就是Generator函数
   5. This的指向，默认指向类的实例，但是一旦单独使用该方法，就会报错，解决：，在构造方法中绑定this或者使用箭头函数
4. 如果类中的一个方法加上static关键字，表示该方法不会被实例继承
   1. 静态方法中的this指向的是类！！！
   2. 父类的静态方法可以被子类继承，静态方法也可以从super对象上调用
   3. 静态方法可以和非静态方法重名
5. 实例属性的新写法
   1. 实例属性可以定义在类的最顶部 class Point{count = 0}等同于class Point { constructor() {this.\_count = 0;}，这种新写法的好处是，所有实例对象自身的属性都定义在类的头部，看上去比较整齐，一眼就能看出这个类有哪些实例属性。
6. 静态属性指的是class本身的属性，第一种写法，用类名.属性名 = 值；或者在实例属性前加上static关键字（此方法是新的提案）
7. 私有方法和私有属性
   1. 如何变为私有方法
      1. 方法名以\_开头
      2. 将私有方法移除模块，定义一个function，在类内部调用的时候要改变移除的方法的this指向
      3. 将私有方法的命名为一个Symbol值
   2. 私有属性提案，是在属性前加#
8. New.target；该属性一般用在构造函数之中，返回new命令作用于的那个构造函数，如果构造函数不是通过new命令或Reflect.construct()调用的，new.target会返回undefined，
9. **Class的继承（重点内容）**
10. 通过extends关键字实现继承，父类的静态方法也会被子类继承
11. 子类在继承的时候，必须要在constructor中调用super（表示父类的构造函数）方法，否则会报错，因为子类自己的this对象，必须要先通过父类的构造函数进行塑造，得要与父类相同的实例属性和方法。如果不调用super方法，子类得不到this对象。只有super方法才能调用父类实例
12. Es5的继承，是通过创造子类的实例对象this，然后再将父类的方法添加到this。ES6则是先将父类的方法和属性，添加到this上，然后再用子类的构造函数修改this
13. Super关键字：
    1. 既可以当做函数，也可以当做对象使用
       1. 作为函数调用时，代表父类的构造函数，Super虽然代表父类的构造函数，但是this指向的是子类的实例，即super内部的this指向的是子类的实例。子类中的this代表父类.prototype.constructor.call(this)作为函数时，只能在子类的构造函数中使用
       2. 当做对象时，在普通方法中，指向父类的原型对象，在静态方法中，指向父类。在普通方法中，由于super指向的是父类的原型对象，所以定义在父类实例上的属性或方法是无法通过super调用的，只能调用定义在原型对象上的属性或者方法，并且，在子类普通方法中使用super调用父类的方法时，方法中的this指向的是子类的实例。在子类的静态方法中通过super调用父类的方法时，方法内部的this指向当前的子类，而不是子类的实例。
    2. 使用super的时候，必须显示指定是作为函数还是作为对象使用，否则会报错
    3. 对象总是继承其他对象的，所以可以在任意一个对象中，使用super关键字。
14. 类的 prototype 属性和\_\_proto\_\_属性
    1. Class 作为构造函数的语法糖，同时有prototype属性和\_\_proto\_\_属性，因此同时存在两条继承链；作为一个对象，子类（B）的原型（\_\_proto\_\_属性）是父类（A）；作为一个构造函数，子类（B）的原型对象（prototype属性）是父类的原型对象（prototype属性）的实例。
       1. 子类的\_\_proto\_\_属性，表示构造函数的继承，总是指向父类
       2. 子类prototype属性的\_\_proto\_\_属性，表示方法的继承，总是指向父类的prototype属性
15. 实例的\_\_proto\_\_属性
    1. 子类实例的\_\_proto\_\_属性的\_\_proto\_\_属性，指向父类实例的\_\_proto\_\_属性。也就是说，子类的原型的原型，是父类的原型。
16. Es6允许继承原生构造函数，因为 ES6 是先新建父类的实例对象this，然后再用子类的构造函数修饰this，使得父类的所有行为都可以继承
17. Mixin 模式，指的是多个对象合成一个新的对象，新对象具有各个组成成员的接口