## ES6相关

### 1.es6熟悉吗，说几个es6的新增的一些东西

1. 新增声明命令let和const
2. 模板字符串（Template String）
3. 函数的扩展(默认参数和箭头函数)
4. 对象扩展
5. import和export
6. Promise
7. 解构赋值
8. 展开运算符(...运算符)
9. Reflection反射

### 2. let count的详解

1. let 和 const 都是块级作用域。以{}代码块作为作用域范围 只能在代码块里面使用。
2. 不存在变量提升，只能先声明再使用，否则会报错。在代码块内，在声明变量之前，该变量都是不可用的。这在语法上，称为“暂时性死区”（temporal dead zone，简称 TDZ）。
3. 在同一个代码块内，不允许重复声明。
4. const声明的是一个只读常量，在声明时就需要赋值。（如果 const 的是一个对象，对象所包含的值是可以被修改的。抽象一点儿说，就是对象所指向的地址不能改变，而变量成员是可以修改的。）

### 3. 模板字符串详解

用一对反引号(`)标识，它可以当作普通字符串使用，也可以用来定义多行字符串，也可以在字符串中嵌入变量，js表达式或函数，变量、js表达式或函数需要写在${ }中

var str = `abc  
def  
gh`;  
console.log(str);  
let name = "小明";  
function a() {  
 return "ming";  
}  
console.log(`我的名字叫做${name}，年龄${17+2}岁，性别${'男'}，游戏ID：${a()}`);

### 4. 函数的扩展详解

1. 可以给函数参数设置默认值

function A(a,b=1){  
 console.log(a+b);  
}  
A(1); //2  
A(2,3); //5

1. 箭头函数
   1. 可以省略function关键字
   2. 可以不写函数名
   3. 如果只有一个表达式可以不需要写{} 和 返回值

//省略写法  
var people = name => 'hello' + name;  
   
var getFullName = (firstName, lastName) => {  
 var fullName = firstName + lastName;  
 return fullName;  
}

### 5. 对象的扩展详解

1. 属性的简写。ES6 允许在对象之中，直接写变量。这时，属性名为变量名, 属性值为变量的值
2. 方法简写

var o = {  
 method() {  
 return "Hello!";  
 }  
};  
   
// 等同于  
var o = {  
 method: function() {  
 return "Hello!";  
 }  
};

1. Object.keys()方法，获取对象的所有属性名或方法名（不包括原形的内容），返回一个数组

var obj={name: "john", age: "21", getName: function () { alert(this.name)}};  
console.log(Object.keys(obj)); // ["name", "age", "getName"]  
console.log(Object.keys(obj).length); //3  
   
console.log(Object.keys(["aa", "bb", "cc"])); //["0", "1", "2"]  
console.log(Object.keys("abcdef")); //["0", "1", "2", "3", "4", "5"]

1. Object.assign ()，assign方法将多个原对象的属性和方法都合并到了目标对象上面。可以接收多个参数，第一个参数是目标对象，后面的都是源对象

var target = {}; //目标对象  
var source1 = {name : 'ming', age: '19'}; //源对象1  
var source2 = {sex : '女'}; //源对象2  
var source3 = {sex : '男'}; //源对象3，和source2中的对象有同名属性sex  
Object.assign(target,source1,source2,source3);  
   
console.log(target); //{name : 'ming', age: '19', sex: '男'}

### 6. import和export模块化引包和导出详解

1. export用于对外输出本模块（一个文件可以理解为一个模块）变量的接口。
2. import用于在一个模块中加载另一个含有export接口的模块。
3. import和export命令只能在模块的顶部，不能在代码块之中

//导入部分  
//全部导入  
import Person from './example'  
   
//将整个模块所有导出内容当做单一对象，用as起别名  
import \* as example from "./example.js"  
console.log(example.name)  
console.log(example.getName())  
   
//导入部分  
import { name } from './example'  
   
//导出部分  
// 导出默认  
export default App  
   
// 部分导出  
export class User extend Component {};

### 7. Promise对象异步对象

1. Promise的概念和作用： Promise是异步编程的一种解决方案，将异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数
2. Promise的状态： 它有三种状态，分别是pending进行中、resolved已完成、rejected已失败
3. Promise语法： Promise 构造函数包含一个参数和一个带有 resolve（解析）和 reject（拒绝）两个参数的回调。在回调中执行一些操作（例如异步），如果一切都正常，则调用 resolve，否则调用 reject。对于已经实例化过的 promise 对象可以调用 promise.then() 方法，传递 resolve 和 reject 方法作为回调。then()方法接收两个参数：onResolve和onReject，分别代表当前 promise 对象在成功或失败时

语法：

var promise = new Promise((resolve, reject) => {  
 var success = true;  
 if (success) {  
 resolve('成功');  
 } else {  
 reject('失败');  
 }  
}).then(  
 (data) => { console.log(data)},  
 (data) => { console.log(data)}  
)

执行过程：

setTimeout(function() {  
 console.log(0);  
}, 0);  
var promise = new Promise((resolve, reject) => {  
 console.log(1);  
 setTimeout(function () {  
 var success = true;  
 if (success) {  
 resolve('成功');  
 } else {  
 reject('失败');  
 }  
 },2000);  
}).then(  
 (data) => { console.log(data)},  
 (data) => { console.log(data)}  
);  
console.log(promise); //<pending> 进行中  
setTimeout(function () {  
 console.log(promise); //<resolved> 已完成  
},2500);  
console.log(2);  
   
//1  
//Promise {<pending>}  
//2  
//0  
//成功  
//Promise {<resolved>: undefined}

### 8. 解构赋值详解

ES6 允许按照一定模式，从数组和对象中提取值，对变量进行赋值，这被称为解构（Destructuring）

1. 数组的解构赋值 数组中的值会自动被解析到对应接收该值的变量中，数组的解构赋值要一一对应 如果有对应不上的就是undefined

* var [name, pwd, sex]=["小周", "123456", "男"];  
  console.log(name) //小周  
  console.log(pwd)//123456  
  console.log(sex)//男

1. 对象的解构赋值 对象的解构与数组有一个重要的不同。数组的元素是按次序排列的，变量的取值由它的位置决定；而对象的属性没有次序，变量必须与属性同名，才能取到正确的值

* var obj={name:"小周", pwd:"123456", sex:"男"}  
  var {name, pwd, sex}=obj;  
  console.log(name) //小周  
  console.log(pwd)//123456  
  console.log(sex)//男  
    
  //如果想要变量名和属性名不同，要写成这样  
  let { foo: foz, bar: baz } = { foo: "aaa", bar: "bbb" };  
  console.log(foz) // "aaa"  
  console.log(foo) // error: foo is not defined

### 9. 展开运算符(...运算符)

1. 将字符串转成数组

* var str="abcd";  
  console.log([...str]) // ["a", "b", "c", "d"]

1. 将集合转成数组

* var sets=new Set([1,2,3,4,5])  
  console.log([...sets]) // [1, 2, 3, 4, 5]

1. 两个数组的合并

* var a1=[1,2,3];  
  var a2=[4,5,6];  
  console.log([...a1,...a2]); //[1, 2, 3, 4, 5, 6]

1. 在函数中，用来代替arguments参数 ```js rest参数  …变量名称

rest 参数是一个数组 ，它的后面不能再有参数，不然会报错

function func(...args){ console.log(args);//[1, 2, 3, 4] } func(1, 2, 3, 4);

function f(x, ...y) { console.log(x); console.log(y); } f('a', 'b', 'c'); //a 和 ["b","c"] f('a') //a 和 [] f() //undefined ```

### 10. 反射 Reflection

1. 为什么需要反射 ?很多强类型语言长期以来都有其反射(Reflection)API（如 Python 或 C#），而 JavaScript 作为一种动态语言，则几乎用不着反射。在 ES6 特性里引入的少量扩展之处中，允许开发者用Proxy访问此前的一些语言内部行为就算得上一项。你可能会反驳，尽管在规范和社区中没有明确那么称呼过，但 JS 在 ES5 中已经有反射特性了。诸如 Array.isArray, Object.getOwnPropertyDescriptor, 甚至 Object.keys 这些，在其他语言中都是典型的被列为反射的方法。而内置的 Reflect 对象则更进了一步，将这些方法归纳在一起。这很有用，是吧？为什么要用超类 Object的静态反射方法(如getOwnPropertyDescriptor或 create)呢？毕竟Object表示一个基本原型更合适，而不是成为反射方法的仓库。用一个专有接口暴露更多反射方法更有意义。
2. Reflect的语法 返回值 vs 通过 Object 反射

// 和 Object 中等价的 Reflect 反射方法同时也提供了更有意义的返回值。比如，Reflect.defineProperty方法返回一个布尔值，表示属性是否被成功定义；而对应的Object.defineProperty则返回其首个参数中接收到的对象 -- 这并不是很有用。举例来说，以下代码演示了Object.defineProperty如何工作  
  
try {  
 Object.defineProperty(target, 'foo', { value: 'bar' })  
 // yay!  
} catch (e) {  
 // oops.  
}  
// 相反，用Reflect.defineProperty就会感觉自然得多：  
var yay = Reflect.defineProperty(target, 'foo', { value: 'bar' })  
if (yay) {  
 // yay!  
} else {  
 // oops.  
}  
// 这种方法免去了使用try/catch 代码块，并使得代码更易维护。