一、let 和 const

```
1. let
  1. 用来声名变量
  2. for循环中 i只在本轮循环有效，所以可以实现for+let 实现闭包
  3. 在声名之前是不可用的，叫‘暂时性死区’
    typeof x; // ReferenceError
    let x;
    --------------
    typeof undeclared_variable // "undefined"
  4. let不允许在相同作用域内，重复声明同一个变量。
  5. 不能在函数内部重新声明参数
      function func(arg) {
        let arg; // 报错
      }

      function func(arg) {
        {
          let arg; // 不报错
        }
      }

      let b=1;  window.b//undefined
2. const 用来声名常量
  1. 不允许重新赋值
  2. const一旦声明变量，必须立即初始化，不能留到以后赋值
  3. const保证变量指向的内存不可改动，而不是声明的值不能改动
```

相同点

```
1. 不存在变量提升
2. 只在当前作用域内有用
3. 不允许重复声名（不允许在相同作用域内，重复声明）
4. ES6 开始,es6全局变量将逐步与顶层对象的属性脱钩。
  let b = 1;
  window.b // undefined
```

const和let形成的影响

es6怎么实现es5闭包

```
改成var
ES6 let语法实现闭包，let 允许你声明一个作用域被限制在块级中的变量、语句或者表达式
  for(let i=0;i<10;i++){
        setTimeout(function() {
            console.log(i)
        }, 1000*i);
    }
```

什么时候用var什么用let

不建议使用var　let、const在任何情况都优于var
因为var定义的变量没有块级作用域，还会出现变量提升的情况，，这样会导致意想不到的错误

es6声明变量的6种方法

var,function,const,let,import,class
7种数据类型：undefined、null、布尔值（Boolean）、字符串（String）、数值（Number）、对象（Object）
　新的数据类型　symbol(字符串)

undefind null boolean array string number object

二、字符串和数组的扩展、set

字符串的扩展

1. 模板字符串　　`Hello ${name}, how are you ${time}?`
2. for...of 遍历字符串
3. .includes(), startsWith(), endsWith()
   includes()：返回布尔值，表示是否找到了参数字符串。
   startsWith()：返回布尔值，表示参数字符串是否在原字符串的头部。
   endsWith()：返回布尔值，表示参数字符串是否在原字符串的尾部
4. repeates() 将字符串重复几次　　str.repeats(3)
   1. 小数会被取 2.9=>2 -0.9=>0
   2. 参数为负数或infanity　会报错

数组

1. Array.from
   1. 伪数组转换为数组;
   2. 将字符串分割成数组
   3. 将Set结构的数据转换为真正的数组：
   4. Array.from参数是一个真正的数组：Array.from会返回一个一模一样的新数组(可以实现深拷贝(不同的对象))
2. Array.of 将一组值转换为数组　　Array.of(3, 11, 8)
3. 扩展运算符　...[1, 2, 3]　（1，2，3）
   应用求数组中的最大值:Math.max(...[2,5,8]) --> Math.max(2,5,8)

   将数组push到数组的后面
   let a = [1,2];
   a.push(...[3,4])
   -->a.push(3); a.push(4)

   将字符串转化为数组
   　　[...'hello']
   // ["h", "e", "l", "l", "o"]
4. 数组实例的 copyWithin()　　Array.prototype.copyWithin(target, start = 0, end = this.length)
   1. [1, 2, 3, 4, 5].copyWithin(0, 3)//[4,5,3,4,5]
5. 数组实例的 find() 和 findIndex()
   [1, 4, -5, 10].find((n) => n < 0)　//-5
   [1, 5, 10, 15].findIndex(function(value, index, arr) {

```
    return value > 9;
  }) // 2
6. 数组实例的 fill()
  1. ['a', 'b', 'c'].fill(7) //[7,7,7]
  2. ['a', 'b', 'c'].fill(7, 1, 2)(填充的数字,start,end)
7. 数组实例的 entries()，keys() 和 values()
  keys()是对键名的遍历、values()是对键值的遍历，entries()是对键值对的遍历。
    for (let index of ['a', 'b'].keys()) {
      console.log(index);//0   1
    }
8. 数组实例的 includes()
9. 数组的空位
```

set数据结构

```
1.新的数据结构 Set。它类似于数组，但是成员的值都是唯一的，没有重复的值。
```

解构赋值

...[1,2,3] console.log(1,2,3)

三、异步编程

实现异步编程的四种方法（http://www.ruanyifeng.com/blog/2012/12/asynchronous _ javascript.html)

```
1. 回调函数
  用setTimeout({callback()},times)
2. 事件监听
  f1.on('done', f2);当f1发生done事件，就执行f2
  function f1(){

       setTimeout(function () {
          // f1的任务代码
          f1.trigger('done');
        }, 1000);
     }
3. 发布/订阅
    jQuery.subscribe("done", f2);f2向"信号中心"jQuery订阅"done"信号。
    function f1(){
           setTimeout(function () {
              // f1的任务代码
              jQuery.publish("done");
           }, 1000);
       }
  jQuery.unsubscribe("done", f2);f2完成执行后，也可以取消订阅（unsubscribe）。
4. Promises对象

var p=new Promise(function(resolve,reject){
  console.log()
    resolve()
    })

  Promise
  .all([runAsync1(), runAsync2(), runAsync3()])
```

```
  .then(function(results){
      console.log(results);
  });  //[results1,results2,result3]

  Promise
  .race([runAsync1(), runAsync2(), runAsync3()])
  .then(function(results){
      console.log(results);
  });//谁反应最快返回谁
```

Promise

```
1. 缺点：
  1. 无法取消promise
  2. 如不设置回调函数，promise内部抛出的错误，无法反应到外部
  3. 当处于pending状态时，无法得知目前进展到哪一个阶段
2. 优点
  1. 异步的操作以同步的流程表达出来
  2. 避层层嵌套回调
3. new Promise(()
4. Promise.prototype.finally() 不管最后状态如何，都会执行的操作
5. p=Promise.all([]).then().catch()用于将多个Promise实例，包装成一个新的Promise实例,全部状态变成fulfilled,p的状态才会变成fulfilled进入then
6. p=promise.race([]) 率先改变状态， p也跟着改变
7. promise.resolve() 将现有对象转成Promise对象
  const jsPromise = Promise.resolve($.ajax('/whatever.json'));
  将 jQuery 生成的deferred对象，转为一个新的 Promise 对象。
8. Promise.reject（）返回的也是一个promise
```

地狱回调

```
1. 什么是错误优先的回调函数？
  错误优先的回调函数用于同时返回错误和数据，第一个参数返回错误，并且验证他是否出错其他参数用于返回数据
  fs.readFile(filePath, function(err, data){
      if (err){
          // 处理错误
          return console.log(err);
      }
      console.log(data);
  });

2. 如何避免地狱回调
  1. 模块化：将回调函数转换为独立的函数
  2. promise
  3. 使用aync/await
  4. 使用流程控制库


4. 可以使用监控unhandledRejection来补货所有未处理的Promise错误
```

手写ppomise

2. 定义resolve reject函数
3. 加入延时机制，setTimeout 0,
4. 加入三种状态：pending，fulfilled,rejected
5. 链式promise:then方法中，创建并返回新的promise
6. 失败处理
7. 异常处理

四、 箭头函数

ES6中箭头函数中this

```
（1）函数体内的this对象，就是定义时所在的对象，而不是使用时所在的对象。
（2）不可以当作构造函数，也就是说，不可以使用new命令，否则会抛出一个错误。
（3）不可以使用arguments对象，该对象在函数体内不存在。如果要用，可以用Rest参数代替。
（4）不可以使用yield命令，因此箭头函数不能用作Generator函数。
  var data = {
       a: 1,
       b: function () {
           return this.a;
       },
       c: this.a,
       d: () => {
           return this;
       },
       e: (function () {
           return this;
       })
   }
   console.log(data.b());
   //属于对象的方法调用:所以this是data对象
   console.log(data.c);
   //js中只有函数能生成作用域,函数意外的地方this都指向window:所以是undefined
   console.log(data.d());
   //箭头函数中的this都指向其定义的环境中的this:由于箭头函数外部就是全局环境
   // 所以this指向window
   console.log(data.e());
//此时箭头函数的外部是一个对象的方法,所以this指向方法内部的this,所以是data对象
```

箭头函数的this和普通函数，

```
1. 普通函数中this:
  1. 总是指向直接调用者， obj.func ,那么func中的this就是obj
  2. 在非严格模式下，没找到直接调用者，就是window
  3. 严格模式下，没有直接调用者，this就是undefined
  4. call,apply,bind指向绑定的对象
2. 箭头函数中的this：指向定义时所处的对象；解决了this的指向问题
3. settimeout:window
```

2.单线程的js实现异步

1. ES6之前
    回调函数
    setTimeOut
2. Promise --- 异步对象

Generator

1. generator函数 异步编程解决方案
  1. 是一个状态机，封装多个内部状态
  2. 是一个遍历器对象生成函数，会返回一个遍历器对象；
2. 特征
  1. function 与函数名间有一个* ；
  2. 内部使用yield表达式，定义内部的内部状态
3. Generator函数调用，调用后，该函数并不执行，返回的也不是函数运行结果，而是一个指向内部状态的指针对象，必须调用next方法；
4. Generator函数是分段执行的，yield表达式是暂停执行的标记，而next方法可以恢复执行
5. yield表达式如果在另一个表打死中，必须放在圆括号里面console.log（'hello'+(yield) 3）

6 fal

axync函数:

通过then调用时碰到await就返回一个promise对象,等执行完回调才会执行下面的逻辑
Generator的语法糖
1. 多个await命令，携程继发关系，比较耗时，第一个完成了才能执行下一个；
  let foo = await getFoo(); let foo = await getFoo();
   可以写成:
  let [foo,bar]=await Promise.all([getFoo(),getBar()])

五、class

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

      toString() {
          return '(' + this.x + ', ' + this.y + ')';
      }
  }
```

extends

```
  class A {}
    class B extends A {
        constructor() {
            super();
        }
    }
```

注意，super虽然代表了父类A的构造函数，但是返回的是子类B的实例，即super内部的this指的是B，因此super()在这里
相当于A.prototype.constructor.call(this)。

对象，class，原型有什么区别，

```
构造函数：创建某个类的对象的函数
类是拥有相同属性的对象
自有属性：对象自身的属性
共有属性：对象原型里的属性

对象自身的属性和方法只对该对象有效，而原型链的属性方法对所有实例有效
class:都有一个constructor(构造函数)，用来构造自有属性
        共有属性放在constructor外面就可以了
        实例属性，写入在constructor中
        静态属性与方法，写在constructor外，用static修饰
        原型对象，写在constructor外
        没有原型灵活，但是也有优点：例如让有些属性只读，get age(){return this.age}
```

ES6的运用

es6写一个工厂函数