this 指向问题

答:this 代表当前上下文（环境）对象
1.函数内 this 指向默认 window 对象;严格模式为 undefined
2.加 new 关键字，表示新建一个对象 this 切换为新创建的 Object 对象
3.call 或 apply 改变 this 指向--用于对象的继承--对象的属性继承
--(1)调用函数
--(2)改变函数中 this 指向(通常用来实现继承)
函数对象.call(要改变的对象,实参,实参,...);
函数对象.apply(要改变的对象,[实参,实参,...]);
4.事件处理函数中的 this，当触发事件时，this 改变成当前触发事件的 DOM 对象
5.bind 改变 this 指向 函数对象.bind(要改变的对象),不调用函数，仅改变函数调用时的 this 指向。

箭头函数和 ES5 中函数区别:

1.箭头函数是匿名函数，没有 function 关键字，不能作为构造函数，不能使用 new
2.箭头函数不绑定 arguments，使用 rest 参数...解决
3.箭头函数不绑定 this，会捕获其所在的上下文的 this 值，作为自己的 this 值，自带 bind(this)。
4.箭头函数通过 call()或 apply() 方法调用一个函数时，只传入了一个参数，对 this 并没有影响。
5.箭头函数没有原型属性
6.箭头函数不能当做 Generator 函数,不能使用 yield 关键字

总结
箭头函数的 this 永远指向其上下文的 this，任何方法都改变不了其指向，如 call() ,bind() , apply()
普通函数的 this 指向调用它的那个对象

ES6 新增语法

解构赋值

对象：let {x,y,z} = {x:100,y:200,z:300} // let {x,...rest} = {x:100,y:200,z:300}
数组：let [x,y,z] = [1,2,3,4,5,6] // let [x,...rest] = [1,2,3,4,5,6]

不定参数　　--形参

```
function add( ...values ){
    console.log( values ); // [10, 20, 30]
    let sum = 0;
    for( let i = 0, len = values.length; i < len; i++ ){
        sum += values[i];
    }
    return sum;
}
var res = add( 10, 20, 30 );
console.log( res );   //60
```

拓展参数　　--实参

```
let arr = [ 1, 3, 0, -1, 20, 100 ];
console.log( Math.max( ...arr ) ); //100 ...: 把数组展开
console.log( Math.max( ...arr, 1000 ) ); //1000
```

proxy　　ex6 新增的语法

原生：　对象的代理，get 拦截对象，set 自定义对象

```
let pro = new Proxy({
    name: "Tom",
    sex: "男",
    age: 19,
    Info: function(){
        alert("name",this.name);
    }
    },{
    get:function(target,key){
        console.log("get 方法被调用了！ ");
        return target[key];
    },
    set:function(target,key,value){
        console.log("set 方法被调用了！ ");
        target[key] = value;
    }
});
```

Webpack 配置中：

```
proxy: {
    "/data": { //地址
```

```
            “target”: “http://www.bjlink32.com/data.php”, //接口地址,跨域访问
            // secure: false,//  如果是 https 接口，需要配置这个参数
            "changeOrigin": true,//开启跨域
            "pathRewrite": { "^/data" : "" }//如果接口本身没有/data 需要通过 pathRewrite
        来重写了地址
    }
```

内置对象新增 API

String 新增 API

includes();返回值是布尔值，表示是否包含指定字符串
startsWidth();返回值是布尔值，表示指定字符串是否在字符串的头部
endsWidth();返回值是布尔值，表示指定字符串是否在字符串的尾部
repeat 方法返回一个新字符串，表示将原字符串重复 n 次
新增模板字符串： 代码:demo57.html

```
window.onload=function(){
    var basket={
        count:40,
        onSale:20
    }
    var result=document.getElementById("result");
    result.append(`
        There are ${basket.count} items
        in your basket, ${basket.onSale} are on sale!
    `);
};
// 普通字符串
    let str1=`好好学习，'\n'天天向上`;
    console.log(str1);
// 多行字符串
    let str2=` 态度决定一切,
              教育改变生活,
              天天向上
              `;
    console.log(str2);
// 字符串中嵌入变量
    let name = "Bob", time = "today";
    let str3=`Hello ${name}, how are you ${time}?`;
    console.log(str3);
所有模板字符串的空格和换行，都是被保留的
```

Number 新增 API

javascript 中浮点数计算会产生误差，为了 解决这个问题 ES6 引入了 Number.EPSILON 值是 2.220446049250313e-16 ，这个非常小的值，为浮点数计算设定一个误差范围，误差小于这个范围，我们认为得到了正确结果。

判断是否是整数 Number.isInteger()

Math.trunc()方法用于去除一个数的小数部分，返回整数部分。

Math.sign()方法用来判断一个数到底是正数、负数、还是零。

它会返回五种值。

参数为正数，返回+1;　参数为负数，返回-1;

参数为 0，返回 0;　参数为-0，返回-0;

其他值，返回 NaN。

Array 新增 API

Array.from 将类似组的对象转为真正的数组，补充：[...arr]将数组展开在合并新的数组

只要是部署了 Iterator 接口的数据结构,Array.from()都能将其转为数组。

Array.of()方法用于将一组值，转换为数组

数组实例的 Array.find()方法，用于找出第一个符合条件的数组成员。它的参数是一个回调函数，所有数组成员依次执行该回调函数，直到找出第一个返回值为 true 的成员，然后返回该成员。如果没有符合条件的成员，则返回 undefined 。

数组实例的 find()方法的用法与 findIndex()方法非常类似，返回第一个符合条件的数组成员的位置，如果所有成员都不符合条件，则返回-1 。

fill()方法使用给定值，填充一个数组。

ES6 提供三个新的方法——entries()、keys()和 values() ，用于遍历数组。它们都返回一个遍历器对象，可以用 for of 循环进行遍历，唯一的区别是 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 es6 遍历数组的方法中不会跳出的方法

forEach 返回 undefined 内部有 break 会报错

ES6 新增加的数据结构

1. Set

--值是唯一的

--只有值没有键

--set 是一个没重复值的数组

--用途：一般用于数组的去重，set(arr)返回值返回的是类数组。

2. WeakSet

--值是唯一的

--只有值，没有键

--值必须是对象
--WeakSet 弱引用，无法遍历(可用于垃圾回收)

3. Map

--值-值结构(比 object 范围更广)　　直值对
--特点：属性名可以是任意数据类型
--作用：属性名也可以存值。

4. WeakMap

--类似 Map(只能对象作为键名)

Promise

概述：Promise 是异步操作，　Promise 是一个对象，可以获取异步操作的消息
优点：（1）、避免回调地狱的问题（2）、Promise 对象提供了简洁的 API，使得控制异步操作更加容易
缺点：①语义化不明显，大量使用.then 方法会使代码臃肿。②promise 一但创建无法取消，在 pendding 状态无法得知处于哪一个状态，③如果不设置回调函数，内部的错误无法反应到外部。
Promise 有三种状态：pendding //正在请求，rejected //失败，resolved //成功
基础用法：new Promise(function(resolve,reject){ })
resolved,rejected 函数：在异步事件状态 pendding->resolved 回调成功时，通过调用 resolved 函数返回结果；当异步操作失败时，回调用 rejected 函数显示错误信息
then 的用法：then 中传了两个参数，第一个对应 resolve 的回调，第二个对应 reject 的回调
catch 方法：捕捉 promise 错误函数，和 then 函数参数中 rejected 作用一样，处理错误，由于 Promise 抛出错误具有冒泡性质，能够不断传递，会传到 catch 中，所以一般来说所有错误处理放在 catch 中，then 中只处理成功的，同时 catch 还会捕捉 resolved 中抛出的异常

all 方法：Promise.all([promise1,promise2])——参数是对象数组。
以慢为准，等数组中所有的 promise 对象状态为 resolved 时，该对象就为 resolved；只要数组中有任意一个 promise 对象状态为 rejected，该对象就为 rejected
race 方法：Promise.race([promise1,promise2])——参数是对象数组。以快为准，数组中所有的 promise 对象，有一个先执行了何种状态，该对象就为何种状态，并执行相应函数

promise 和 fetch 方法可以取消吗？

Promise 取消的方法:
第一种:使用 rejected
第二种:使用 promiser。Race　竞速方法<
原生 promise 有个缺点就是没有一种能真正取消 fetch 请求的方式，直到最近这个问题才得到解决。一个新的 AbortController 添加到了 JavaScript 规范里，它让我们可以使用一个信号来终止一个或者多个 fetch 请求。

AbortController 读音：　　饿 抱 t ，啃抽乐

Promise　.then 的第二个回调函数

then 的作用是为 Promise 实例添加状态改变时的回调函数。then 方法的第一个参数是 resolved 状态的回调函数，第二个参数（可选）是 rejected 状态的回调函数。

then 方法返回的是一个新的 Promise 实例（注意，不是原来那个 Promise 实例）。因此可以采用链式写法，即 then 方法后面再调用另一个 then 方法。

```
getJSON("/post/1.json").then(function(post) {
return getJSON(post.commentURL);
}).then(function funcA(comments) {
console.log("resolved: ", comments);
}, function funcB(err){
console.log("rejected: ", err);
});
```

上面代码中，第一个 then 方法指定的回调函数，返回的是另一个 Promise 对象。这时，第二个 then 方法指定的回调函数，就会等待这个新 Promise 对象的状态发生变化。如果新 promise 状态变为 resolved，就调用第一个参数，如果状态变为 rejected，就调用第二个参数，后边有 catch 就会被 catch 接收并抛出错误。

Super 的作用

（1）Es6 中可以通过 super 调用父类的构造器函数，继承属性
（2）super 的另外一个作用是调用父类的 protected 函数。只有通过"super"，才能操作父类的 protected 成员别无它法。Super 相当于 es5 中 –父类函数名.call(this)改变 this 指向，

调用父类的方法。

es6 如何实现异步 --generator 函数

async/await：
async 是 Generator 函数的语法糖，async 用于声明一个 function 是异步的，await 用于等待一个异步方法执行完成。
async 函数的优点：（1）内置执行器（2）语义化更好（3）更广的适用性

地狱回调怎么解决？

1. 可以拆解 function，将各步拆解为多个 function
a. Promise 使得多个嵌套的异步调用能够通过链式的 API 进行操作
b. Generator
调用 Generator 函数，返回一个遍历器对象（Iterator）
调用遍历器对象的 next() 方法，就会返回一个有着 value 和 done 两个属性的对象
c. async 表示这是一个 async 函数，await 只能用在这个函数里面
await 表示在这里等待 Promise 返回结果后，再继续执行
await 后面跟着的应该是一个 Promise 对象（当然，其他返回值也没关系，只是会立即执行）
await 等待的虽然是 Promise 对象，但不必写 .then() ，可以直接得到返回值

es6 严格模式的语法要求

不用 var 声明变量，也不允许函数重名，新增的关键字不能当成标示符使用，并且函数声明必须在顶层。arguments 被定义为参数，不用在于其他参数绑定。with、caller、八进制法、所有 eval 的操作都被禁用。给对象的只读属性进行赋值、还有禁止拓展的对象添加新属性时、删除系统内置属性的时候都会报错。

Var let const 的区别

let const 和 var 区别:
(1)let const 块作用域 {} ；而 var 是函数作用域
(2)let const 都不同重复定义；而 var 可以重复定义
let 和 const 区别:

let 创建一个块级变量，后面可以重新赋值

const 创建一个块级常量，后面不可以重新赋值

如果定义一个空对象或数组用 let 还是 const，想改变里面的值　为什么？

const 实际上保证的并不是变量的值不得改动，而是变量指向的那个内存地址所保存的数据不得改动

对于赋值数据类型来说，用 const 定义内部的值可以更改，但是不能重新赋值

const 定义对象属性是否能修改？（为什么属性可以修改）

对象是引用类型的，const 定义的对象中保存的是指向对象的指针，这里的“不变”指的是指向对象的指针不变，而修改对象中的属性并不会让指向对象的指针发生变化，也就是对象的地址不变所以用 const 定义对象，对象的属性是可以改变的

3.删除对象里的某一种属性？

delete 可以删除对象中的对象属性，但是构造器原型上的属性不能删除

解构/rest 解构

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，叫解构

对象：let {x,y,z} = {x:100,y:200,z:300}　//　let {x,...rest} = {x:100,y:200,z:300}

数组：let [x,y,z] = [1,2,3,4,5,6]　//　let [x,...rest] = [1,2,3,4,5,6]

异步：yield / generator / promise / async/await

Yield：

使用 promise 封装一个异步操作

原生 js ajax-->模仿 jquery ajax 封装-->ES6 Promise-->jQuery ajax 封装成 Promise 版本

--fetch promise 版本的 ajax 封装

--axios 原生 js 中 ajax 封装

ES6 Promise 异步对象作用-->改写异步封装-->改写 ajax 异步封装

---可读性强

---易于维护

```
$.ajax({})      --pending
.then(function(data){})      --resolve
.catch(function(e){})    --reject
axios()      --pending
.then(function(data){}) --resolve
.catch(function(e){})    --reject
```

call/apply/bind 区别

在 JS 中，这三者都是用来改变函数的 this 对象的指向的，

call 方法第一个参数是要绑定给 this 的值，后面传入的是一个参数列表。当第一个参数为 null、undefined 的时候，默认指向 window。

apply 接受两个参数，第一个参数是要绑定给 this 的值，第二个参数是一个参数数组。当第一个参数为 null、undefined 的时候，默认指向 window。

bind 第一个参数是 this 的指向，从第二个参数开始是接收的参数列表。区别在于 bind 方法返回值是函数以及 bind 接收的参数列表的使用。bind 方法不会立即执行，而是返回一个改变了上下文 this 后的函数。

(ES6 的箭头函数下，call 和 apply 将失效，对于箭头函数来说:

箭头函数体内的 this 对象，就是定义时所在的对象，而不是使用时所在的对象;

所以不需要类似于 var _this = this 这种丑陋的写法

箭头函数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

箭头函数不可以使用 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

不可以使用 yield 命令，因此箭头函数不能用作 Generator 函数，什么是 Generator 函数可自行)

Generator

函数相当于一个状态机，　函数调用后返回一个遍历器对象（iterator），我们可以通过使用遍历器对象的 next 方法，来遍历每一个 yield 状态

yield 语句的返回值　　--undefined
yield 语句后面表达式的值或函数调用的返回值
遍历器对象.
next()返回值 value:值，done: true/ false}

yield 语句后面表达式的返回值作为 next 方法返回值的 value 值

.ES5,ES6 怎么实现继承？

a. ES5:构造函数继承（call/apply/bind）
　　原型链继承：　子构造函数.prototype = new 父构造函数()
　　组合式继承：　构造函数+原型链继承
b. ES6: 主用使用 class 使用关键字 extends
　　子类必须在 constructor 方法中调用 super 方法

ES6 新增加的数据结构

--Set/Map/Weakset/WeakMap
--Symbol 基础数据类型，是独一无二的值
--作用：在开发中，为了避免变量名冲突。

proxy 拦截器—代理

ES6 原生提供 Proxy 构造函数，用来生成 Proxy 实例。Proxy 这个词的原意是代理，用在这里表示由它来“代理”某些操作，可以译为“代理器”。就是在目标对象之前架设一层“拦截”，外界对该对象的访问，都必须先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。

Async：

async 函数返回一个 Promise 对象
async 函数内部 return 语句返回的值，会成为 then 方法回调函数的参数
async 函数内部抛出错误，会导致返回的 Promise 对象变为 reject 状态，抛出的错误对象会被 catch 方法回调函数接收到
4、async 函数有多种使用形式

Await：

正常情况下，await 命令后面是一个 Promise 对象。如果不是，会被转成一个立即 resolve 的 Promise 对象。
await 命令后面的 Promise 对象如果变为 reject 状态，则 reject 的参数会被 catch 方法的回调函数接收到
Promise 对象的状态变化：async 函数返回的 Promise 对象，必须等到内部所有 await 命令后面的 Promise 对象执行完，才会发生状态改变，除非遇到 return 语句或者抛出错误。也就是说，只有 async 函数内部的异步操作执行完，才会执行 then 方法指定的回调函数
只要一个 await 语句后面的 Promise 变为 reject，那么整个 async 函数都会中断执行
前一个异步操作失败，也不要中断后面的异步操作。这时可以将第一个 await 放在 try...catch 结构里面，这样不管这个异步操作是否成功，第二个 await 都会执行
错误处理：如果 await 后面的异步操作出错，那么等同于 async 函数返回的 Promise 对象被 reject
多个 await 命令后面的异步操作，如果不存在继发关系，最好让它们同时触发

async/await？一个普通的函数和 async 修饰的函数有什么区别？

async/await 是一种建立在 Promise 之上的编写异步或非阻塞代码的新方法
async 是 ES7 中用于修饰函数，被修饰的函数返回一个 Promise 对象
普通函数有 return 的就返回 return 后面的表达式，没有 return 的返回 undefined

拓展参数、不定参数

扩展参数 //扩展参数是另一种形式的语法糖，它允许传递数组或者类数组直接作为函数的参数而不通过 apply

不定参数 //不定参数就是在函数中使用命名参数同时接收不定数量的未命名参数

现在有十个接口，等十个接口全部返回然后在做页面的渲染，这十个接口有可能会出现网络错误，用 promise 的哪个 API？（错误的信息和正确的信息都要渲染到页面上）

Promise.all 方法
可以将多个 promise 对象做链接。