一、第一部分： JavaScript

一、JavaScript 的基础语法

1.JS语言的组成

1536724100060

2.书写位置：

有三种：

1. Script 的src 链接外部js文件
2. 写在script标签中 没有src属性
3. 写在【 】点击a 链接执行的js代码

3.声明变量及赋值（重点）

1. 如何声明变量（容器）： 通过关键字var
2. 对变量赋值
3. 如何同时声明变量及赋值
4. 同时声明多个变量

   补充命名规则：**变量的命名规则：数字、字母、下划线、（$） [不能以数字开头、不能使用关键字或保留字，严格区分大小写] 【命名建议： 见名知意、驼峰命名规则】

4.数据类型

1. 数字类型 number var a=10;
2. 字符串类型 String var a=”qzz”;
3. 布尔类型 boolean true/false； var tianqi=true

转义字符：\

5.输出方法：

1. alert（） 弹窗输出
2. Document.write()；在页面中写出；往body里面输出内容
3. Console.log() 在控制台输出，多用于调试代码

如何进行字符串和变量的拼接（字符串的拼接）？？？

1. 字符串拼接：当加号两侧只要有一侧为字符串，那么代表字符串拼接 【引号引号，加号加号，引号里面加号，加号里面加内】

获取文本框的内容的值、通过value

7.数据类型的转换：

【文本框获取的值是字符串类型，若进行加法运算，会变成字符串拼接】

1. 直接转换

(1) Number（）转换为数字 【结果：数字、NaN】 NaN代表非纯数字

(2) String（） 转换成字符串

(3) Boolean（）转换成布尔值【除了 0、空字符串、NaN、null、undefined 之外都是true】

2. 隐式转换

(1) 加法不可以进行隐式转换

(2) 当隐式转换后，运算仍无法进行，会得到NaN

9. 字符串的比较：

NaN： 非数值。NaN不等于NaN 【isNaN():判断是不是数字】

常用方法：取整、取浮点数

number.toFiexed(n)保留n位小数

Math.random() 生成[0,1)的随机数，包含0，不包含1 【parseInt(Math.random()*101)】:生成0~100的随机整；

报错：在元素结构产生前，先获取元素会报错，因为代码执行顺序是从上到下的

10.运算符

1.关系运算：

== 和 != 和 === 【判断当两边的值不仅内容，数据类型完全相等时才

为true 】 !==【判断当两边的值不仅内容，数据类型不完全相等时才

为true

隐式转换后也相等。

2.逻辑运算：

1. && 逻辑与【运算符的两侧都为true的时候，最终返回true，短路原则：左侧返回false右侧将不执行】
2. || 逻辑或 【】

操作符的优先级：

“（）” > 一元运算符 > 算术运算符 > 关系运算符(大于小于高于等于) > 逻辑运算符（&&>||） > 赋值运算符

复习：

Ele.value ==> 只有表单元素有的属性

11.进制转换：

1. 二进制：0b开头

2. 八进制：0o开头

3. 十进制：

4. 十六进制：ox开头

12.如何进行进制的转换：

1. 十进制转换 【number=100; number.toString(16);转成16进制；toString(8);转成8进制;number是要转换的值】

2. 多进制转换十进制：

```
var x = "100";  转换成10进制parseInt(x,2);  parseInt("",多少进制);    parseInt("f",16) --> 15
```

13.特殊数据类型：

1. null 空对象 数据类型是NULL

2. undefined 数据类型是Undefined； 代表变量声明了，未被赋值；【区分报错： not defined 代表变量未声明】

14.数据类型的判断：typeof()

Typeof() 方法

```
typeof(null);//"object"    typeof(typeof 12)//"string"  typeof("number")//string
```

15.顺序执行、选择分支语句、循环语句

1.**判断语句：if switch**

A. If语句：

1. 单分支if(){}

2. 双分支if(){}else{}

3. 多分支if(){} else if(){}..... else if(){} else{}

B. Switch 语句 【只有switch才有break 语句】

比较值的时候用的是全等于“===” 只有值的内容和数据类型完全相等的时候

switch(变量){

case 值1： ... break;

...

default: 所有条件都不满足的时候，才执行这里的代码；（break）

```
switch(true){

    Case 关系表达式：  ...  break;

}
```

2.**循环语句：**

三大要素：变量初始化、条件、变量更新；

注意：编写条件的时，要避免出现死循环；

1. while语句：

(1) 变量初始化

```
(2)      While(条件){

         条件满足时，执行这里的js的代码；

         变量更新；

      }

     【while(true){  }】

(3) 注：

            水仙花数：153 = 1*1*1+5*5*5+3*3*3
```

2.for循环

3.do...while循环

Break 和continue 区别：

Break：跳出循环；后面的代码将不在执行

Continue:跳出本次循环，进行下一次循环

如果要打破外层循环，需要加标识符：【标识符名：循环{ 循环{ break 标识符名 } } 】 EX：

补：若break和continue 后带标识，则跳出标识所在循环

16 . 三元运算：条件？条件成立：条件不成立；

补充：获取select ： 自己选中的option

```
var mySel = document.getElementById("sel");//获取select的ID

        var index  =mySel.selectedIndex ;//获取自己选中的option

        var marks = mySel.options[index].value; //获取自己选中的option的值
```

二、JS深入

1.函数

1.函数的声明0

(1) **关键字声明： function 函数名（形参）{ 执行的js代码 }**

(2) **赋值式声明： var变量（函数名） = function （形参）{ 执行的js代码 }**

(3) **构造函数 new Function() { }**

重复声明函数（同名的函数）：后面的会覆盖前面的（声明提前）

声明提前的概念：

① **【在执行JS代码前，会将所有声明提升到**当前作用域的最前面】**

② **从上到下按照程序的三大流程执行代码：**

【undefined： 变量声明但未赋值】

③ **注意：不能在赋值式定义函数前，执行函数，会报错 ** is not a function 【原因是声明提前，此时变量为 undefined ，是一个值】**

④ **以下这种方法： 是可以的 ，在关键字声明函数前，执行函数**

【因为会new Function这样一个对象】

⑤

2. 函数的执行

(1) **函数名() 【去调用这个函数并执行它】**

(2) **事件驱动 onclick="函数名 ( ) " 【写在HTML标签中，一般不建议这样使用，注意结构样式行为三分离】**

用法： 元素.onclick = 函数名 【这里不加括号】

3. 函数的分类：

① **内置函数 ： 无需自己定义，就能去用**

② **自定义函数：自己定义的**

③ **匿名函数：没有名字的函数 btn.onclick = function(){ JS执行的代码 }**

(4)

补：随机数：

Math.random()*(max-min+1)+min; 取min 到max的随机数

4.作用域：

（当在某个函数内使用某个变量）

1. 全局作用域：（全局变量）：在函数最外层声明的变量

2. 局部作用域（局部变量）：在函数内声明的变量

作用域链：当函数访问变量时，根据**就近原则**从**内到外**查询变量，这个路径称为作用域链。

5.变量的访问规则：

【当前函数-->父级函数-->继续上层-->直到最顶层-->not defined】

就近原则（如查找变量a）：

* 使用变量a时先从**当前函数**查找，如果有则可以使用;

* 如果当前函数无变量a,则往**父级函数**查找，如果找到则使用，并停止查找;

* 如果在父级函数还是无法找到，则继续**往上一层函数**查找，以此类推;

* 直到**最顶层(全局作用域)**，如果还是没找到，则报**错误 ** is not defined;**

6.函数的形参与实参：

作用：将函数外部的值传入函数内部

形参：函数定义时的参数（变量）

实参：函数执行时的参数（值） 实参是**基本数据类型**相当于传递的是值

```
fn(a);  //是实参
```

【arguments】类数组：保存实参的信息

索引：

7.函数的返回值 【return】

作用：

1. 将函数内部的值返回到函数外部；外部需要应用变量接受该值。若函数没有书写return，函数默认的返回值为undefined；

2. 跳出函数，终止函数的执行

3. Return 后面的代码将不执行，只能跳出一层函数

8. 函数的this

当前对象，指的是调用函数的对象，函数大部分手动执行时，this都只是window

（谁调用函数，this指向的就是谁）事件驱动执行的话，this指向驱动事件的那个元素对象

9.递归函数：

函数自己调用自己，容易进入死循环，例如

2. 数组：

【不确定数组的长度，栈中的内存比较小，数组在栈中存放堆中的地址】

1. 在JavaScript中，数组中的每个元素的类型是可以不同的；

【是因为JavaScript的弱数据类型决定数组中元素类型可以不同的； 因为在JavaScript中，定义变量的时候不指定其数据类型，仅仅用一个var来表示当前对象是一个变量，至于其是什么类型的不指定，在后面使用的时候可以赋值不同的数值类型。】

1.数组创建的方式：(2种)

1. 字面量： var arr = [1,2,3..]
2. 使用构造函数创建 var arr = new Array(); var arr = new Array(4); 创建一个长度为4的数组
3. var arr = new Array[“123”,”223”,...] 创建数组时并对元素赋值
4. 【带数值与不带数值的】

2.常用的数组方法：9种 （重点）

（都是小写）

数组的增删改查，数组的索引 a[i]

1. 对数组中的元素进行赋值（长度不是固定，如：赋初值的时候，只有3个值，也可以操作a[5]）

1.push()：

往数组尾部添加一个或多个元素，返回数组新的长度

2.pop()：

删除数组最后一个元素，返回删除的元素

3.unshift()：

往数组开头添加一个或多个元素，**返回数组新的长度**

4.shift()：

删除数组第一个元素，返回删除的元素

5.splice(start,deleteNum,...items)：

在数组中插入、删除、替换的通用方法

start：起始索引位置

deleteNum：要删除的数量

items：插入的元素（可以是多个）

删除：arr.splice(2,2)

修改：arr.splice(1,1,"qinzz");

6.slice(start[,end])：

返回数组的片段或子数组，从start开始到end(不包括end所对应的元素)

如果省略end参数，则截取到数组的最后一项,支持负数

7.sort()：

将数组中的元素排序，并返回排序后的数组

默认:以字符串的排列方式（转换成ASCII码进行对比）

参数fn（比较函数）：利用fn的返回值来确定数组中两个数的位置（假设两个数为a,b）

```
function fn(a,b){
    return a>b; //默认从小到大  a<b是从大到小
}
arr.sort(fn());
```

返回负数：确定a在b前，[a,b...]

返回0：不改变现有位置

返回正数：确定a在b后面，[b,a...]

补充：对象的排序：

```
goodLists.sort(function(obj1,obj2){
            var val1 = obj1.date;
            var val2 = obj2.date;
            if (val1 < val2) {
                return -1;
            } else if (val1 > val2) {
                return 1;
            } else {
                return 0;
            }
        });
```

8.reverse()：

将数组中的元素颠倒顺序，返回逆序后的数组

9.join(separator)

返回字符串值，其中包含了连接到一起的数组的所有元素【常用 .join("")】

separator为分隔符，不写就默认为逗号

10.reduce() 实现数组的累加 用在斐波那契数列

###

3.字符串：

一、字符串的创建

1．字面量 var str = “”;

2．构造函数 var str = new String(“”);

二、字符串的操作

1．通过索引获取字符串的某个字符【str[索引值] --> 只能在es5用】【str.charAt(索引值)】

2．字符串的长度 str.length 只读

3．获取某个字符在字符串中的索引 str.indexOf(“字符”) 若没有该字符，则返回值是 -1；

三、字符串的其他方法

1．replace(str|regExp,”)

【替换字符串】

这里的替换只能执行一次，不能够进行全局匹配，如果需要全局匹配，则应使用正则表达式

g: 全局变量

i: 不区分大小写

2. split("分割符")

字符串的分割方法,根据分割字符，把字符串拆分成数组

4. 数组对象排序

1. 如果数组项是对象，我们需要根据数组项的某个属性对数组进行排序，要怎么办呢？其实和前面的比较函数也差不多：

```
var arr = [{name: "zlw", age: 24}, {name: "wlz", age: 25}];
var compare = function (obj1, obj2) {
    var val1 = obj1.name;
    var val2 = obj2.name;
    if (val1 < val2) {
        return -1;
    } else if (val1 > val2) {
        return 1;
    } else {
        return 0;
    }
}
console.log(arr.sort(compare));
```

5.时间对象： （时间对象的创建)

1.不带参数：

返回代码执行的时间（本地的时间）

```
var date = new Date();
```

2.带参数：

返回的是具体的日期

(1) 数字：var d = new Date(20162134) 【数字：距离1970-01-01的毫秒数】

(2) 字符串：var d = new Date(“2020/09/20 12:04:09”) //具体的日期时间有固定的格式【1997-02-18 或 1998/09/17】

3.转化日期格式的方法：

1.时间对象的获取方法

(1) getFullYear() 返回的是对象的年份

(2) getMonth() 返回的是对象的月份 返回0-11，代表1-12

(3) getDate() 返回的是对象的日期

(4) getDay() 返回的是对象的星期几，返回0-6，代表星期日到星期六

(5) getHours() 返回的是小时数

(6) getMinutes() 返回的是分钟数

(7) getSeconds() 返回的是秒数

2.时间和日期的补0操作

```

```

4.时间对象的设置方法

【无法设置星期数】

```
        // setFullYear() 改变的是对象的年份
        // setMonth() 改变的是对象的月份。0-11，分别代表一到十二月
        // setDate() 改变的是对象的日
        // setHours() 改变的是小时数
        // setMinutes() 改变的是分钟数
        // setSeconds() 改变的是秒数
```

new Date（obj.time).getTime(); 【转成时间对象】

5. 日期处理【判断两个日期的相差天数】

(1) 获取毫秒数 getTime()

(2) 设置毫秒数 setTime()

Es5: Date的方法（不是日期对象的方法）

```
var seconds = (Date.parse("2018-10-01 00:00:00") - Date.now())/1000;
       var tian = parseInt(seconds/60/60/24);
       console.log(tian);

       es5方法获取毫秒数:【Date的方法（不是日期对象d的方法）】
       // Date.parse(“2015-08-24”)//返回指定日期距1970-1-1零时的毫秒数
       //PS：转换格式默认支持2015-08-24或2015/08/24
       //Date.now();//返回执行这行代码时距1970-1-1零时的毫秒数
```

6.定时器

```
setTimeout(fn,200)：两百毫秒后执行fn这个函数（只执行一次）,返回一个id标识
clearTimeout(timeoutID)：清除指定id标识的延迟操作
setInterval(fn,30)：每隔30毫秒执行一次fn这个函数,返回一个id标识
clearInterval(intervalID)：清除指定id标识的定时器操作
```

1.setInterval(fn,time)

每隔多少毫秒执行fn代码，当定时器执行的时候才执行的fn代码

【fn:要执行的函数[不加括号] time：每间隔多少毫秒执行fn】

要注意的是：当用事件开启定时器之前要注意清空上一次的定时器，尽量将定时器名字声明为函数外

```
    btn.onclick = function(){
            clearInterval(timer);
            //定时器
            timer = setInterval(function(){
                num++;
                console.log(num);
            }, 1000)
        }
```

2. seTimeout(fn,time)

只执行了一次，延迟多少毫秒执行fn代码 ,用法与setInterval相同

3.clearInterval(定时器的名字)

1.clearInterval(定时器的名字) : 用来清除setInterval（fn,time）定时器的 2.clearTimeout(定时器的名字)： 用来清除setTimeout (fn,time) 定时器的

7. window对象

1.window对象定义和要注意的点

(1) 最顶端的对象

(2) 省略了调用对象的方法，其实都是window的方法【尽量不要对属性及事件，省略window对象的调用】

(3) 定义在全局环境下的变量都会成为window对象的属性【若函数内没有用var声明变量，也是全局变量，也 是window的属性】

① 命名冲突② 全局污染

(4) 全局变量起名字要避免window已存在的命名（name,top）

(5) 不要对window属性进行赋值【ex:window.innerHTML = 500,以后获取浏览器的可视化区域都为500】

(6) 通过var声明的全局变量，无法用delete删除

2.window对象的属性及方法

1.常用属性

```
1. window.scrollX  /  window.scrollY  滚动条滚动的距离
2. window.innerWidth / window.innerHeight // 浏览器窗口可视化的宽度
3. outerWidth/outerHeight //表示整个浏览器窗口的尺寸
```

2.常用方法

```
1. window.scrollTo(x,y) ;   设置浏览器当前的滚动距离

2.window.scrollBy(x,y) ;   设置浏览器基于当前浏览器的滚动距离，可以去负值（向下???向上？？？）
```

```
3.document对象的方法也是window对象的

4.alert()            会停止代码的执行

5.confirm(msg)//弹出警告框，返回布尔值，       会停止代码的执行

6.prompt(msg,default)//弹出输入框，返回消息或null ，      会停止代码的执行

7.open(url,name,[options]) : 打开一个新窗口并返回新 window 对象

    name:不命名会每次打开新窗口，命名的第一次打开新窗口,之后在这个窗口中加载

    options为字符串：`'width=400,height=400,top=200,left=200'`

    opener父窗口对象，通过open方法打开的窗口才有opener对象

8. close(): 关闭窗口

9. print(): 调出打印对话框
```

3.属性对象

1.属性对象：

【既是window属性，也是对象】

```
//document 文档对象模型
    // 2.history 包含窗口的浏览历史
    //  * back() 加载 history 列表中的前一个 URL。
    //  * forward() 加载 history 列表中的下一个 URL。
    //  * go() 加载 history 列表中的某个具体页面，支持负数。
    //  * length 返回浏览器历史列表中的 URL 数量。

    console.dir(window.document);
```

2.location（重点）

当前窗口中加载文档的相关信息。 * hash 设置或返回从井号 (#) 开始的 URL（锚）==>哈希值。 * href 设置或返回完整的 URL。

* location.href="[文件夹/]aa.html?name=lemon&age=17"

*search 设置或返回从问号 (?) 开始的 URL（查询部分）, 用location.search 接收到的对象是下面这样的格式，需要用的时候需要进行字符串的裁剪 str.split("=")

```
?g_name=test1Name&g_price=89.99&
```

* 参数的作用：传递信息给另一个页面，告知具体显示内容

```
        * ?name=lemon&age=17
```

补充：This.getAttribute(“属性名”)【获取自定义HTML属性值】

【最好手动转码】

3.document 文档对象模型

4.案例 JS实现div在屏幕居中，页面尺寸改变的时候也居中

【当页面尺寸改变的时候触发的事件 window.onresize() 】

补： 如何获取body 及 html

document.body

document.documentElement

5.利用节点关系、获取其他节点：

(包含元素节点、文本节点)

1. 节点的属性 nodeType

【 元素节点的nodeType是1，文本节点的nodeType是3，属性节点的nodeType是2 】

2. 操作元素的类名 ele.className = "";

8.Dom对象：

1.获取元素节点：

```
// 1. document.getElementById()  通过id获取元素，返回值为元素节点(对象)或者为null(空对象)
        //      * 必须通过document调用
        //      * 速度最快
        //  2. getElementsByClassName() 通过类名获取元素,返回类数组，如果类名不存在返回空数组[]
        //     * 元素节点均可调用
        //  3. getElementsByTagName() 通过标签名获取元素,返回类数组，如果类名不存在返回空数组[]
        //     * 元素节点均可调用
        //  4. document.getElementsByName() 通过name获取元素,返回类数组，如果类名不存在返回空数组[]
        //      *必须通过document调用
```

2.利用节点关系获取其他节点

（文本、元素）

1. ele.parentNode 获取父元素的节点
2. ele.childNodes 获取ele节点的所有自己节点，是数组

3. firstChild
4. lastChild
5. nextSibling 获得节点的下一个兄弟节点
6. previousSibling 得到节点的上一个兄弟节点

3.元素节点的获取，常用：

1. ele.parentElement 得到父元素的节点
2. ele.children 获取到所有的子元素节点
3. ele.firstElementChild
4. ele.lastElementChild
5. ele.nextElementSibling 获取到ele的下一个兄弟节点
6. ele.previousElementSibling

封装实现过滤，只获取元素节点

```

```

4.节点的三大属性：nodeType、nodeName、nodeValue

nodeType【节点的类型：文本返回值3；属性返回值2；元素的返回值1；】

nodeName 【节点名字：返回节点的名字： 文本（#text）/ 属性（属性名字）/ 元素（标签名大写）】

nodeValue【文本：返回文本内容，属性：属性值，元素：很多空格和null】

6、重点：节点的增删改查

1．节点的创建与插入

（1）创建

指定的元素节点：**Documnet.createElement(**“标签名”);**

比如：Documnet.createElement(“h3”)

创建文本节点： Documnet.createTextNode(“啦啦啦”); 【了解，不常用，用innerHTML 也是一样的】

（2）插入

节点的插入：**parent.appendChild(ele)** 给父元素添加**最后**一个子元素

```
window.onload = function(){
    var h3 = documnet.createElement("h3);
    document.body.appendChild(h3);
}
```

2.节点、节点属性的操作

（1）创建属性节点

```
var a =  Documnet.createAttribute("id");  //获取元素节点
a.nodeValue = "id名";   //给节点的一个值
h3.setAttributeNode("")   //设置节点属性  如h3.id = "wangcai"
```

（2）在指定的节点插入

【insertBeefore】

```
var hh = document.getElementsByTagName("div")[1];
var xx = document.getcreateElement("div");
xx.innerHTML = "lalalala";
document.body.insertBefore(xx,hh);
```

（3）节点的删除

parent.removeChild(); (常用)

（4）节点的复制

jiedian.cloneNode(Boolean复制程度)【取值：false 浅复制（只能复制当前行无法复制里面的内容，不常用），true 深复制：这个标签里的所有的东西都会被复制】 复制之后还需要插入，一般用

父元素.appendChild(子元素);

【补充： this 当前执行 onclick 函数的对象，谁执行就是谁】

7.节点的属性和方法：

1．DOM操作标准属性：HTML

键盘相关的属性：

1.which 【判断某个键被按下】

2.兼容性 var keyCode = e.which || e.keyCode

2.节点的属性和方法

（一）节点的属性 * 所有的html结构的标准html属性均可作为节点的属性，DOM节点均可通过.或 [] 获取 【变量一定要用方括号获取" [] "】 * className 类名 * tagName 标签名 * innerHTML * innerWidth

(二) 节点的方法（利用该方法获取html结构的所有html属性） getAttribute("html属性") 获取html元素的属性 setAttribute("html属性","html属性值") 设置html元素的属性

(三) 盒模型相关的节点属性

```
        // 获取元素的宽高（包含content+padding+border）
                offsetWidth/offsetHeight
        // 获取元素到最近的定位父辈(或者html)的距离
                offsetLeft/offsetTop
```

(四) 获取元素的样式

1. window.getComputedStyle(ele节点) 返回值为包含所有css样式的对象（标准浏览器） * window.getComputedStyle(ele节点).css属性 返回值为css属性值

2. ele节点.currentStyle.css属性 返回值为css属性值

不常用，一般都用 ele.style.属性名 = “ ”;

注意事项：ie浏览器都不能直接获取css总属性的值

3.案列：tab标签切换

思路： 1）初始化 * 高亮第一个tab * 隐藏除第一张以外的图片 2）切换：鼠标点击tab（关键：获取点击的index值） * 高亮显示当前tab,去除其他所有高亮（遍历） * 切换相应的图片，隐藏其他所有图片（遍历）

9.浏览器的默认行为与阻止

浏览器的默认行为

1．a标签自动跳转

2．拖拽鼠标文字被选中

3．Submit自动提交

4．右键菜单

阻止

标准浏览器 e.preventDefault()

IE8-: e.returnValue = false

9.事件

1.事件的方法

【 如：input.onclick = function(){ } 】

onclick 当用户点击某个对象时调用的事件。 ondblclick 当用户双击某个对象时调用的事件。

onmousedown 鼠标按钮被按下。 onmouseup 鼠标按键被松开。

onmouseover 鼠标移到某元素之上。 onmouseout 鼠标从某元素移开。

onmousemove 鼠标被移动时触发。

* onclick=onmousedown+onmouseup * ondblclick = onclick *2

2.event对象

监听事件执行过程中的状态，用来保存当前事件的信息对象 e.什么去调用

(一) event对象的鼠标属性

event.button = 0：鼠标左键 / 1：鼠标滚轮 / 2：鼠标右键 1. button 返回当事件被触发时，哪个鼠标按钮被点击。 * 0-1-2

（二）光标相关的属性 clientX /clientY 光标相对于浏览器可视区域的位置，也就是浏览器坐标。 screenX/screenY 光标指针相对于电脑屏幕的水平/垂直坐标。 pageX/pageY:鼠标相对于文档的位置。 * 包括滚动条滚动的距离，即：e.pageX = e.clientX+window.scrollX * IE8-不支持 offsetX,offsetY: 光标相对于事件源对象的相对偏移量。 * 事件源对象：触发事件的对象

（三）键盘相关属性

onkeydown 某个键盘按键被按下。 onkeyup 某个键盘按键被松开。 onkeypress 键盘<字符键>被按下触发,而且如果按住不放的话，会重复触发此事件。 字母、数字、空格、标点符号、换行

event对象的键盘属性 // 1. which 返回当事件被触发时，哪个键盘按键被点击。 // * 对于keydown和keyup事件，它指定了被敲击的键的虚拟键盘码。 // * 对于keypress事件，该属性声明了被敲击的键生成的 Unicode 字符码(ascii码) // * 左上右下 37 38 39 40

3.事件冒泡

概念:对象上触发某类事件，那么事件就会沿着DOM树向父级传播，从里到外，直至它被处理程序处理，或者事件到达了最顶层（document/window）【从下往上】

阻止事件冒泡的方法：

```
e.stopPropagation();
e.cancelBubble = true;  //IE
兼容写法 e.stopPropagation?e.stopPropagation() : e.cancelBubble = true;
```

4.事件委托

1.概念：利用冒泡原理，将自己的执行函数委托给父元素进行执行 2.影响程序执行效率的操作： （1）绑定过多事件 （2）频繁操作dom节点 (3) 请求次数 3.事件源对象：触发事件的元素 【e.target：当前触发事件处理程序的对象 】 标准属性：target IE8-属性：srcElement

兼容写法： var target = e.target || e.srcElement;

【复习：

一、事件 二、事件对象（事件处理函数的参数） 概念：保存事件信息 兼容写法：e = e || window.event; （一）鼠标属性 button 判断按下的是鼠标的哪个键 clientX、clientY 光标到浏览器可视区域的距离 screenX、screenY 光标到屏幕的距离 pageX、pageY 光标到文档的距离(包含滚动条滚动过的距离) e.pageX = e.clientX + window.scrollX offsetX、offsetY 光标到事件源对象的距离 (二) 键盘属性 which 判断按下是哪个键盘的键 ctrlKey 判断有没有按下ctrl键，返回布尔值 altKey 判断有没有按下alt键，返回布尔值 shiftlKey 判断有没有按下shift键，返回布尔值 //组合键 e.ctrlKey && e.which == 90 三、事件冒泡 * 事件委托 * 事件源对象 * 兼容写法：var target = e.target || e.srcElement; * 阻止冒泡 * 兼容写法：e.stopPropagation? e.stopPropagation() : e.cancelBubble = true;】

5.事件的绑定方式：

（1）DOM节点绑定 ele.on+type = fn; ex: ele.onclick = function(){ } * 无法设置捕获阶段 * 同一节点的同一事件会被覆盖

（2）作为html属性 （3）事件监听器 ele.addEventListener(click， fn, 是否捕获); * 可以设置捕获阶段 * 可以给同一节点设置多个同一事件 标准： ele.addEventListener(type,fn,isCapture) * type事件类型 * fn 事件处理函数 * isCapture 是否捕获，默认为false冒泡。 ie写法 : ele.attachEvent(on+type,fn) * ie不支持捕获阶段 二、事件操作 * 事件冒泡 * 事件捕获 每个事件都只能在冒泡或捕获阶段执行一次 执行同一事件时，先捕获再冒泡。

6.事件的移除

1. dom节点： ele.on+type=null 2. 事件监听器： ele.removeEventListener(type,fn) * 移除事件时，type事件类型一致，fn是同一个函数，才可以移除。 IE的写法： ele.detachEvent(type,fn)

10.Cookie（重点）

一、cookie的基本设置及获取

* 客户端与服务器端进行通讯使用的，一个能够在浏览器本地化存储的技术。 * 设置 document.cookie = "name=value"; * 每一次都只能设置一条cookie * 获取 document.cookie * 返回值的格式为字符串"name=laoxie; left=283; top=229" * 一次性获取到所有cookie

二、cookie的其他参数

1.设置： document.cookie = "name=value[; expires=date][; path=/]" * expires 有效期 * session 默认为临时存储 * date 具体日期的字符串 toUTCString() * path 保存cookie的位置 * 默认当前文件所在目录 * / 存储到根目录

三、JSON字符串

'{"name":"lemon","price":98.88}';

- 格式：
- 属性名和字符串必须使用双引号
- 不能有注释
- 不能存在多余逗号
- 属性值必须为以下类型 * String * Number * Boolean * Object * Array * Null
- 转换

  对象/数组 -> json字符串： JSON.stringify(数组||对象)

  json字符串 -> 对象/数组： JSON.parse()

  cookie存放的是字符串。现在需求我需要存放数组对象，要使用JSON字符串
- 空字符串无法转换成JSON对象

```
2.获取：document.cookie
    一次性获取到当前目录往上找到根目录的所有cookie
```

11.正则表达式

1.构造函数创建

```
var reg= new RegExp('study');
//使用特殊字符
var reg= new RegExp('\\d\\w+');         \d\w+

var reg = new RegExp('study', 'ig');

var reg = /study/gi;    //使用字面量创建，不支持变量
```

- i: case-insensitive，表示忽略大小写
- g: global，表示全局匹配
- m: multiline，表示多行匹配

2.支持正则表达式的字符串方法

1. str.search(正则) 返回第一次匹配时所在的索引值，如果匹配不到就返回-1
2. str.match() 默认匹配字符串，返回一个数组

   0：所匹配的字符

   index：匹配第一个字符所在的索引

   input: 对字符串的引用

   - 全局匹配(g)，返回一个匹配所有字符串数组
   - 如果匹配不到则返回null

   3.str.replace 替换字符

   .str.split 'a,b,c,d,e'.split(/\s,\s /);

   5. 测试正则：
      1. 格式：正则表达式.test(字符串)
      2. 用<正则表达式>测试<字符串>是否匹配,返回true/false

```
/xx/.exec(字符串)
```

匹配规则

3. 所有字母和数字都是按照字面量进行匹配,和字符串匹配等效 `/good/gi`
4. 字符类（只记小写字母即可）
   - `.` : 除换行以外的字符
   - \w : 代表数字或字母或下划线

- \W : 非数字字母和下划线字符
- \d : 数字
- \D : 非数字
- \s : 代表一个空格
- \S : 空格以外的字符
- \b : 匹配一个单词边界，也就是指单词和空格间的位置
- \B : 匹配非单词边界。

  注： 以上所有的字符类都只是匹配"一个字符

5. 特殊字符

1. ^ $. * + ? = ! : | \ / () [] {}
2. []: 代表任意"单个字符" ,里面的内容表示"或"的关系
- -: 代表范围
- ^: 代表非
3. (): 表示分组（n是以最左边括号出现的顺序排列）
- $1: 表示第一个分组
- $n: 表示第n个分组（不能写在正则表达式里）
- \n: 在正则分组后面使用，表示对第n个分组的引用(一定要写在正则表达式里)

  PS: 编写的正则分组数量越少越好

4. |: 表示或者
5. 锚点定位
- ^: 表示以什么开头
- $: 表示以什么结尾
6. 表示数量，对前一个字符计数，
- *: 代表0个或0个以上 <===>{0,}
- +: 代表1个或1个以上 <===>{1,}
- ?: 代表0个或1个 <===>{0,1}
- {}:

  - ```
    \d{5}: 匹配5个数字
    \d{5,10}: 匹配5个到10个数字
    \d{5,}: 匹配5个或5个以上的数字
    ```

注意贪婪模式和非贪婪模式

PS: 1）数量词*,+,{5,}，会尽可能多的去匹配结果（贪婪） 2）在后面加一个?表示尽可能少的去匹配结果（非贪婪：找到需要的内容就停止匹配） google,goooogle ==> /go+/

var reg = /b\w+?d/g 【获取一个字符串中所有b到d之间的内容】str.match(reg);

三、ES5-ES6
    ```

ES5: IE9以上才支持

一、ES5

1.页面加载事件

【等待页面加载完毕之后触发】

1. window.onload = function(){} 页面所有的加载完毕之后才开始执行【不常用，经常会慢】
2. 页面加载顺序：
    1. 解析HTML结构
    2. 加载外部脚本和样式表文件
    3. 解析并执行脚本代码
    4. DOM树构建完成 【页面加载应该在这一步获取】
    5. 加载图片等外部文件
    6. 页面加载完毕
3. Dom事件加载
    1. onreadystatechange 当页面准备阶段改变的时候触发
        1. 属性：
            1. readyState 页面的准备阶段

                【值：interactive DOM树构建完成后 / complete 页面加载完成后 ，相当于window.onload 】
        2. DOMContentLoaded(只能使用事件监听器) 【当DOM树构建完毕触发】 用的比较多

2.JS的严格模式

1. 了解：es4 改动太大了 所以没有发布
    1. es5 的 严格模式 和 普通模式【按照严格模式写代码】 “use strict" IE9以上才支持
2. 注意严格模式

    使用严格模式的目的是：
    1. 消除javascript语法的一些不合理，不严谨的地方，减少一些怪异行为；
    2. 消除代码运行的一些不安全之处，保证代码运行的安全；
    3. 提高编译器效率，增加运行速度；
    4. 为未来新版本的javascript做好铺垫；

    两种模式：【全局和局部】
    - 全局：针对整个js文件 将”use strict”放在脚本文件的第一行，则整个脚本都将以”严格模式”运行
    - 局部：针对单个函数 将”use strict”放在函数体的第一行，则整个函数以”严格模式”运行。

    执行限制

    1.不使用var声明变量严格模式中将不通过 2.删除系统内置的属性会报错 3.delete不可删除属性的对象时报错，如： 4.var声明的全局变量（会自动称为window的属性） 5.对象有重名的属性将报错 var obj= {name:"小王",name:'王大锤'} 6.函数有重名的参数将报错 function sum(a,a,b){} 7.arguments严格定义为参数 不允许对arguments赋值 禁止使用arguments.callee

8.函数必须声明在顶层，不能写在条件判断语句或for循环语句中

3.获取元素节点

1.document.querySelector(“css选择器”) 只能获取满足css选择器的第一个元素,返回dom节点对象
2.document.querySelectorAll(”css选择器“) 获取满足css选择器的所有元素,返回数组

注：css选择器在css中怎么写，在这里就怎么写

4.操作类的属性

Function方法：

- - bind()
    - 用于将当前函数和指定对象绑定(改变this指向)，返回一个新的函数应用

```
for(var i=0;i<btns.length;i++){
            btns[i].idx = i;
            btns[i].onclick = function(){
                //this:当前绑定事件的对象
                setTimeout(function(){
                    console.log(this);
                }.bind(this),2000)
            }
        }
```

  -

获取class列表属性：

classList：js 的方法

- length : class类名的个数
- add() : 添加class方法（常用）
- remove() : 删除class方法（常用）
- toggle() : 切换class方法
- contains():是否含有某个类,返回布尔值

```
ele.classList.add("active");  //在原来的基础上添加一个类名
```

data自定义属性：

可以设置也可以获取，w3c自定义属性

设置：

获取到的是一个对象：键包含所有的属性

【必须通过set/getAttribute()拿到或者获取】

二、ES6

1.变量声明

- let: 代码块内的变量声明 1）变量声明不会提前 2）块级作用域 {}以内 3）let不允许相同作用域内多次声明同一变量

  1537839713857

- const:常量声明 1）变量声明不会提前 2）块级作用域 3）const不允许相同作用域内多次声明同一变量 4）声明后无法修改值： 不成文的写法：const MY_NAME

  PS：const常用于引用第三方库的声明

2.解构赋值Destructuring

1. ES6允许我们对数组和对象中提取值，对变量进行赋值，这被叫做“解构”；
    1. 数组解构：

    ```
    var arr = [1,"aa","bb",3];
    var [a,b,c,d=0] = arr;
    //当d对应的arr没有值的时候，就是第二条语句里默认的值
    ```

    2. 对象解构

    ```
    var obj = {
                name : "laotian",
                age : 28,
                gender : "girl"
            }
    var {name,age:nianling,gender,hobby="boy"} = obj;
    ```

    若变量名与键名不一致，通过键名：变量名 进行关联；

    若变量名与对象的键名不一致，会解构失败，返回undefined

3.字符串扩展

1. 方法：
    1. str.includes("字符") 【判断字符串str是否存在某个字符】返回布尔值
    2. startsWith/ensWith()【判断字符串是否以某个字符开头或结尾】
    3. repeat(n) 【得到字符串重复n次后的结果，n可以为小数，但不能为负数】（一个字符串复制n次，输出）
2. 字符串模板

使用反引号``表示,通过 ${变量} 进行拼接

4.数组扩展（Set集合）

5.对象扩展

对象的方法：

1. 传递：obj1 = obj2; 传递的是地址
   2. 拷贝
   1. 深拷贝：先变成JSON字符串，在变成JSON对象
   2. 浅拷贝：Object.assign(obj1,obj2) 将所有的对象合并到obj1，并返回obj1 【对象的复制： var newObj = Object.assign({},obj1) 】

6.箭头函数arrow function (重点)

1. 参数与返回值

```
没有参数
//传统写法
var sum = function(){
    return 10 + 10;
}
//es6箭头函数
var sum = () => 10+10;


函数只有一个参数
//传统函数写法
var test = function(x){
    return x+2;
}
//使用箭头函数
var test = x=>x+2;


多个参数
// ES5
var total = values.reduce(function (a, b) {
  return a + b;
}, 0);
// ES6
var total = values.reduce((a, b) => a + b, 0);
```

1. ES6中的规则是，紧随箭头的{被解析为块的开始，而不是对象的开始
2. 使用了块语句的箭头函数不会自动返回值，你需要使用return语句将所需值返回
3. 当使用箭头函数返回一个普通对象时，需要将对象包裹在小括号里
4. 默认参数Default var func1 = (x = 1,y = 2) =>x+y
5. 剩余参数Rest

   var func2 = (x, ...args) => {console.log(args)}; func2(1,2,3); // 输出 [2, 3]

2. 箭头中的this值 【箭头函数没有它自己的this值，箭头函数内的this值继承自外围作用域 】
3.

7.Symbol数据类型（独一无二的值，无法修改）

1. ES6引入了一种新的原始数据类型Symbol，表示独一无二的值，一旦创建后就不可更改，是一种类似于字符串的数据类型，但Symbol 值不能与其他类型的值进行运算，否则报错。
2. Symbol函数可以接受一个字符串作为参数，表示对Symbol实例的描述，主要是为了标识和区分，对调式非常有用 。

```

```

3. 用途
    1. 给对象创建私有属性 给现有的对象添加属性，可能会产生命名冲突，Symbol的出现解决这个问题

```
var mySymbol = Symbol();

// 第一种写法
var a = {};
a[mySymbol] = 'Nani';

// 第二种写法（注意加方括号，否则回被当作普通属性）
var a = {
  [mySymbol]: 'Nani'
};

// 以上写法都得到同样结果
a[mySymbol] // "Nani"
```

    2. 常用方法
        1. Symbol.for()

        有时我们希望重新使用同一个Symbol值，Symbol.for方法可以做到这一点，首先在全局中搜索**已登记**的Symbol值，如果有，就返回这个Symbol值，否则就新建并返回一个以该字符串为名称的Symbol值

```
let one = Symbol("laoxie");
let two = Symbol.for("laoxie");

//由于创建了两个Symbol值，所以他们不相等
console.log(one===two);//false
```

        2.Symbol.keyFor() 获取被登记的Symbol值

        应用：定义私有属性，利用symbol作为键的名字

8.Set集合（可自动去重）

1. Set集合，类似于数组，但是成员的值都是唯一的，可自动去重。
2. 方法：
    1. add(value)：添加某个值，返回Set结构本身。
    2. delete(value)：删除某个值，返回一个布尔值，表示删除是否成功。
    3. has(value)：返回一个布尔值，表示Set集合中是否存在该值。
    4. clear()：清除所有成员，没有返回值。
    5. 去重数组
        1.
```
let items = new Set([1, 2, 3, 4, 5, 5, 5, 5]);

   //去重后从新转成数组
   Array.from(items);
```

```
for...of 方法【只要有迭代器iteractor的地方，都可以使用】

   遍历  【for…of      forEach()】
  var imgs = new Set(['a','b','c']); //根据KEY遍历

       for(let item of imgs){

            console.log(item);

       }

       //a

       //b

       //c
```

```
   imgs.forEach((item,idx)=>{
       console.log(item,idx);
   })
```

```

```

9.Map集合

1. js对象（Object）只能用字符串当作键(属性名)。这让它的使用有了很大的限制。所以ES6推出了一种类似于对象的数据集合：Map集合，它能让所有类型的数据作为属性名 ， 键可以是任意类型
2. 常用方法
    1. 设置set(key, value)
    2. 获取get(key)

3. has(key)
4. delete(key)
5. clear()

```
//创建
let map = new Map();

//设置
map.set("S0", "张三");
map.set("S1", "李四");
map.set("S2", "王五");

//获取
map.get("s2"); //王五
```

3.遍历方法

keys()【获取所有键，返回类数组】

values()【获取所有值，返回类数组】

entries()【获取所有键值对，返回类数组】

forEach()【遍历Map所有成员】

for...of ... 遍历

```
for(let [key,value] of map){
    console.log(key,value);
}
```

10.生成器函数

函数体内部使用了yield 表达式

对象.next() 去执行 【执行next()后得到一个yield 或return返回值组成的对象 {value:xx,done:false} 】【对象中的done是否为true,取决于函数是否完结束】

- function和函数名之间有一个*号
- yield： 暂停代码执行
- return：终止函数
- next()： 执行后得到一个对象，有两个属性如下：
  - value：暂停时返回的值（yield）
  - done：表示函数是否执行完毕

```
function* sum(x){
    console.log(1);
    yield 50;
    console.log(2);
    yield 100;
    console.log(3);
    // return 150;
        console.log('end');
}
var res = sum();//得到一个状态为suspended的对象{next()}
// res.next();//得到一个对象：{value:xx,done:false}
console.log(res);
```

三、Animation动画

1.分类：匀速、加速、减速、抛物线、圆周运动、缓冲运动（先快后慢直到停止）

无缝滚动的思路：

1.复制初始状态下可见区域内的图片并放到最后

2.当复制的图片都滚动到可见区域时，立即把图片定位到初始状态

#

#

==========

#

#

二、第二部分：PHP语法基础

#

一、PHP环境安装

（一）、wampserver安装及使用

1. wampserver安装

- a:安装 Web 服务器Apache
- p:安装 PHP解析器
- m:安装MySQL数据库

> 对于初学者建议使用集成的服务器组件（如：WampServer），它已经包含了 PHP、Apache、Mysql 等服务,免去了开发人员将时间花费在繁琐的配置环境过程。

WampServer下载地址：http://www.wampserver.com/

若是缺乏某个dll，直接安装vc_redist.x64_2015.exe

【计算机-管理-服务和应用程序-服务-wampapache-wampmysql（开启服务前，先停止svn、mysql、iis）】

> 测试：
>
> 当wampserver图片变成绿色：
>
> 1. 网址中输入localhost或127.0.0.1（本机ip），默认打开C:\wamp64\www路径下的index.php
> 2. 同样可以localhost/*** 访问默认路径下（C:\wamp64\www）的任意文件

2.创建虚拟目录

- 左键小图标---->apache---->alias directories--->add an alias
- 创建虚拟目录名字，例如aaa。enter后
- 创建虚拟目录对应的url，即需要开启web服务的文件夹路径(路径不可有中文、空格)
- 以上步骤完毕，访问虚拟目录只需在浏览器输入http://localhost/aaa，可访问目录下任意文件。

3.创建端口

- 左键小图标---->apache---->httpd-vhost.conf
- 复制如下代码，往下写。（//只是注释，记得删掉）
- ```
  listen 1704    //端口号
  <VirtualHost *:1704>      //端口号
          ServerName localhost
          DocumentRoot D:\laoxie\01Basic      //端口对应url(不可有中文路径、空格)
          <Directory  "D:\laoxie\01Basic">    //端口对应url
                  Options +Indexes +Includes +FollowSymLinks +MultiViews
                  AllowOverride All
                  Require all granted     //让所有的主机都可以访问你的服务器
          </Directory>
  </VirtualHost>
  ```

  以上步骤完毕，访问端口下目录只需在浏览器输入http://localhost:1704，可访问目录下任意文件。

4.开启局域网服务器

1.查看本机ip地址：common+r进入cmd，输入ipconfig，查看ipv4。

2.在vhost.conf给端口对应的url添加一句代码

```
#Require local //#代表注释
Require all granted    //允许其他主机访问
```

二、PHP内容

1.概念
```

> PHP是一种通用开源服务端脚本语言，将程序嵌入到HTML文档中去执行，结果以纯 HTML 形式返回给浏览器。【HTML文件可嵌套PHP的代码，但是PHP文件不能嵌套HTML语句】 PHP: Hypertext Preprocessor “超文本预处理器”，1994年由Rasmus Lerdorf创建，刚刚开始仅仅是为了要维护他本人个人网页而制作的一个简单程序（Perl语言编写），原名Personal Home Page（PHP由此得名），后用C语言重新编写，改名Hypertext Preprocessor

PHP能做什么：

- 生成动态页面内容
- 创建、打开、读取、写入文件
- 收集ajax数据
- 发送和接收cookie
- 添加、删除、修改您的数据库中的数据
- 限制用户访问您的网站上的一些页面
- 加密数据

2.基本语法

> - 默认文件扩展名是 “.php”。 【通常放在一个项目的api文件中】
> - 通常包含 HTML 标签和一些 PHP 脚本代码。

分界标示符

```
<?php //开始

//...php代码

?> //结束
```

注释

> 与js一样，分单行和多行注释

- 单行注释：`//`
- 多行注释：`/**/`

输出语句

- **echo**

  只能输出一个或多个字符串（字符串可以包含HTML标签），速度较快，一般用于向前端返回数据

  ```
  <?php
      //输出一个字符
      echo "Hello world!<br>";
      //输出多个字符
      echo "This", " string", " was", " made", " with multiple parameters.";
  ?>
  ```

```
json_encode(array,JSON_UNESCAPED_UNICODE);  把数组转成字符串
        php5.4+ 使用JSON_UNESCAPED_UNICODE防止中文转义
json_decode(json,assoc);把字符串转成数组/对象
        json：待解码的 json string 格式的字符串
        assoc：默认false,返回object，当该参数为 true 时，将返回array
```

- **print_r()**

  打印关于变量的信息，适用于数组、对象的打印，一般用于测试

- **var_dump()**

  判断一个变量的类型与长度,并输出变量的数据类型和数值，一般用于测试

3.变量

命名规则

- 以 $ 符号开始，后面跟着变量的名称（$称为标识符，不属于变量组成部分）
- 只能包含字母、数字字符以及下划线，不能以数字开头（不能包含空格）
- 区分大小写

```
//PHP 没有声明变量的命令，也没有声明提前的概念。
//变量在第一次赋值时被创建：
<?php
    $txt="Hello world!";
?>
```

拼接字符串及变量

```
<?php
    $myname = "lemon";
    echo 'My name is' . $myname;  //1.通过并置运算符.拼接字符串及变量
    //或者
    echo "My name is $myname";  //2.双引号内可以直接输出变量，无需拼接
?>
```

函数中访问全局变量！！！

- 全局变量：在函数外部定义的变量，可以在任意位置访问(需要手动定义为全局变量)
- 局部变量：函数内部声明的变量，仅能在函数内部访问

（1） $GLOBALS

```php
<?php
    $x='global x';
    function myTest(){
        //echo $x;//报错 Undefined variable:未声明定义的变量
        echo $GLOBALS['x']; //1.获取全部变量正确写法：$GLOBALS[变量名]，其中变量名不带$。

        //2.同时，可以在函数中创建全局变量
        $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
    }
    myTest();
    echo $y;
?>
```

（2）global 关键字 【写在别的域里，未出现过的域里】

```php
    <?php
        $x=5;
        function myTest(){
            global $x;   //未出现过的域里
            $y=$x;
        }
        myTest();
        echo $y;
    ?>
```

超级全局变量

- `$GLOBALS`
  是PHP的一个包含所有全局变量的数组，可以在任意位置使用
- `$_SERVER`
  是一个包含了头信息(header)、路径(path)等信息的数组
- `$_POST / $_GET`
  被广泛应用于收集表单数据，常用于ajax请求等操作
- `$_COOKIE`
  用于收集前端发送过来的cookie数据
- `$_REQUEST`
  变量包含了 `$_GET、$_POST 和 $_COOKIE` 的内容
- `$_SESSION`
  服务器版cookie
- `$_FILES`

常量

- 规范
  - 命名规则与变量一致，但常量名不需要加 $ 修饰符。
  - 常量值被定义后，在脚本的其他任何地方都不能被改变。
  - 默认是全局作用域，可以在整个运行的脚本的任何地方使用。
  - 常量名建议全部大写
- 格式： `define(name常量名称,value常量值)`

```
define("EN_NAME", "laoxie");   // const MY_NAME = "lemon";
```

4.运算符及语句（等同于js）

算术运算符、赋值运算符、递增/递减运算符、比较运算符(等于大于...)、逻辑运算符(与或非)、三元运算符 条件语句、循环语句

5.数据类型

$String$（字符串）$Integer$（整型）$Float$（浮点型）$Boolean$（布尔型）$Array$（数组）$Object$（对象）$NULL$（空值）

String

- strlen() 获取字符串长度，得到的字符的字节数
- mb_strlen() 获取字符串长度，
- strpos() 查找某个字符在字符串中的索引，如果未找到匹配，则返回 false

```
strpos("Hello world!","world");//=>6
```

Array

> 数组是一个能在单个变量中存储多个值的特殊变量。

（1）创建数组：array()

```
//1.数值数组，等同于js的数组
$cars=array("Volvo","BMW","Toyota");
//2.关联数组，等同于js的对象
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
//3.多维数组，包含一个或多个数组的数组
```

（2）数组常用方法

- count() 获取数组长度
- in_array() 判断某个值是否存在数组中
- array_slice() 从数组中取出一段【array_slice(要取出的数组名，截取的开始索引，截取的数量[如果不写，默认到最后]) 】
- array_rand() 随机获取数组的索引值

练习案例：在php文件里生成动态商品页面

（3）遍历数组

- for 一般用于遍历数值数组
- foreach() 一般用于遍历关联数组,或者对象【关联数组必须要用foreach(),没有索引只有键值】

```
<?php
    //遍历数值数组：for循环
    $cars=array("Volvo","BMW","Toyota");
    $arrlength=count($cars);
    for($x=0;$x<$arrlength;$x++){
```

```
        echo $cars[$x] . "<br>";
    }
    foreach($age as $item){
        //item代表数组里面每一项的值
    }
    //关联数组:
    $age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
    //获取单个值
    echo $age['peter'];
    //遍历关联数组: foreach..as
    foreach($age as $key=>$value){
        echo $key .":". $value;
        echo "<br>";
    }
?>
```

数组排序

- sort() 对数组进行升序排列
- rsort() 对数组进行降序排列
- asort() 根据关联数组的值，对数组进行升序排列
- ksort() 根据关联数组的键，对数组进行升序排列
- arsort() 根据关联数组的值，对数组进行降序排列
- krsort() 根据关联数组的键，对数组进行降序排列

#

==========

#

#

三、第三部分：Ajax、PHP与MySql的交互

（一）了解AJAX

- AJAX: Asynchronous Javascript And Xml，Ajax 技术的核心是XMLHttpRequest对象（简称XHR），这是由微软首先引入的一个特性，其他浏览器提供商后来都提供了相同的实现

*Ajax起源： 最早出现在2005年的google搜索建议

- ajax优点
    - 增加速度：减轻服务器的负担,按需加载数据,最大程度的减少冗余请求
    - 改善的用户体验：局部刷新页面,减少用户等待时间,带来更好的用户体验
    - 页面和数据分离：前后端分离，操作更灵活，后期维护更方便
- 后端语言和服务器配置
    - php + Apache + mySQL
    - NodeJS + MongoDB
    - Java + tomcat + Oracle

- .NET + IIS + SQL Server

json

json数据(json字符串)

```
{"id" : 21465461461461, "name": "张三"},[{"id" : 21465461461461, "name": "张三"}]
```

json字符串与对象的转换

```
// (一) json字符串转成对象的转换
//1. eval("("+json字符串+")");
它的作用是，将一个普通的json格式的字符串，转换成Json格式的对象
//var list = eval("("+request.responseText+")");
//2. JSON.parse(); //把JSON字符串转成JSON对象(js对象/数组)【es5】

// (二) 把JSON对象转成JSON字符串
JSON.stringify();
```

了解json文件存在的意义

```
//模拟数据(与后端先商量)
[
        {
                "id":"G001",
                "name":"Thermos 膳魔师 Funtainer系列水杯 12盎司 (340g) 粉红色",
                "imgurl":"images/g1.jpg",
                "price":899,
        }
]
```

Ajax请求步骤

创建请求对象,返回一个异步请求对象

```
var xhr = new XMLHttpRequest();
```

处理服务器返回数据

```
xhr.onreadystatechange = function(){
    if(xhr.readyState == 4) {
        //responseText：保存服务器返回的数据（从服务器返回的数据是“字符串”）。
        alert(xhr.responseText);
    }
}
```

设置请求参数，建立与服务器连接

```
xhr.open("get", "http://localhost/api/ajaxtest", true);
```

向服务器发送请求

`xhr.send(null);`

案例：演示向goodslist.json请求数据

XMLHttpRequest对象属性方法

open(type,url（同源策略）,async（同步、异步））

get方法和post方法的区别：

```
1.open(type,url,async)：建立与服务器的连接
    type：请求的类型，get、post
        * 区别?
                get请求数据接在url后面，post请求数据通过send方法传递
                get传递的数据会比较少，post没有限制
                get传递的数据会暴露出来
    url：数据请求的地址（API地址），一般由后端开发人员提供
        * 当前页面访问地址，API地址两者一定要同域
        * 同域（同源策略）：协议，域名，端口三者一致
        * 报错： No 'Access-Control-Allow-Origin' header is present on the requested resource.
Origin 'null' is therefore not allowed access.
        async：是否异步发送请求（true,false），默认为true
        * 同步：按步骤顺序执行，前面的代码执行完后，后面的代码才会执行
                做完前一件事情，才能下一件事情（排队）
        * 异步：与其他操作同时执行，也叫并发（图片加载，ajax请求，定时器）
```

send(data)

```
2.send(data)：向服务器发送请求
        data：可选参数，post请求时才生效，表示发请求时传送的数据字符串。
         xhr.send('size=20&type=music');
                在某些浏览器中，如果不需要通过post请求主体发送数据，则必须传入null
//备注：get请求的数据写在url地址后
        request.open("get", "http://localhost/api/getdata.php?type=get&qty=10", true);
        setRequestHeader(key,val)：设置请求头
//备注：利用请求头设置POST提交数据格式：
        xhr.setRequestHeader('content-type','application/x-www-form-urlencoded');//open方法后设置
```

> **在请求收到服务器的响应后，响应的数据会自动填充xhr 对象的属性，相关的属性简介如下：

readyState

```
0 -  (未初始化) 尚未调用open()方法。
1 -  (启动) 已经调用open()方法，但尚未调用send()方法。
2 -  (发送) send()方法执行完成，但尚未接收到响应。
3 -  (接收) 已经接收到部分响应数据。
4 -  (完成) 响应内容解析完成，可以在客户端调用了
只要readyState 属性的值由一个值变成另一个值，都会触发一次readystatechange 事件。必须在调用open()之前指定onreadystatechange事件处理程序才能确保跨浏览器兼容性。
```

responseText

保存服务器返回的数据（从服务器返回的数据是“字符串”）。

status

```
* 响应的HTTP 状态。
200 (OK) : 服务器成功返回了页面
304 (Not Modified) : 数据与服务器相同，不需要重服务器请求（直接使用缓存的数据）
400 (Bad Request) : 语法错误导致服务器不识别
401 (Unauthorized) : 请求需要用户认证
404 (Not found) : 请求地址不存在
500 (Internal Server Error) : 服务器出错或无响应
503 (ServiceUnavailable) : 由于服务器过载或维护导致无法完
```

（二）、php本地数据操作

获取前端数据

```
$_GET 获取前端用get方式传递过来的数据（url地址后面数据也能被获取）
$_POST  获取前端用get方式传递过来的数据
isset() 判断某个值是否被设置，若不存在返回boolean
```

文件的读取与写入

fopen(path,mode)：打开文件

> 使用fopen函数打开文件时，你首先需要明确打开文件的模式： 1.将数据写入文件，亦或者读写文件 2.考虑如果文件中已经存在相关数据，你是覆盖原有文件中的数据呢，还是仅仅将新数据添加至文件末尾

> 文件模式:
>
> - r 以只读方式打开文件，从文件头开始读
> - r+ 以读写方式打开文件，写入时以追加的方式写入文件
> - w 以写入方式打开文件，从文件头开始写。文件不存在则尝试创建，若存在，则先删除文件中的内容
> - w+ 以读写方式打开文件，从文件头开始读写。文件不存在则尝试创建，若存在，则先删除文件中的内容
> - a 以写入方式打开，从文件末尾开始追加写。如果文件不存在则尝试创建
> - a+ 以读写方式打开，从文件末尾开始追加写入或者读。如果文件不存在则尝试创建。

fread(file,length)：读取内容

fwrite(file,json字符串)：写入内容

fclose(file)：关闭文件,避免资源占用

filesize(path)：读取文件字符长度

```
//1.以读取模式打开文件
$path = './data/weibo.json'
$myfile = fopen($path, 'r');

//2.读取文件内容
$content = fread($myfile, filesize($path));

//3.关闭文件，减少资源占用
fclose($myfile);
```

```
echo返回数据
json_encode(); 把数组转成字符串
        php5.4+ 使用JSON_UNESCAPED_UNICODE防止中文转义
json_decode(json,assoc); 把字符串转成数组/对象
        json：待解码的 json string 格式的字符串
        assoc：默认false,返回嵌套对象的多维数组，通过arr->key获取对象键对应的值。当该参数为 true 时，将返回嵌套关联数组的多维数组，通过arr[key]获取关联数组键对应的值。
```

案例：微博点赞

```
// (html页面) 1.发起ajax请求，将weibo.json的内容返回到页面
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function(){
    if(xhr.readyState === 4 && xhr.status === 200){
        var res = JSON.parse(xhr.responseText);
        ul.innerHTML = res.map(item=>{
            return `<li data-id="${item.id}">
            <h4>${item.username}</h4>
            <p>${item.content}</p>
            <span>${item.comtcnt}</span>
            <span class="like">${item.likecnt}</span>
            </li>`;
        }).join('');
        datalist.appendChild(ul);
    }
}
xhr.open('get','../data/weibo.json',true);
xhr.send();
//(html页面)2.点赞，获取到当前按钮对用的id，发送ajax请求，get方法通过url拼接参数传送id到后台。
//随后php页面对json数据进行修改，再将当前被改变的对象返回。xhr对象通过responseText进行接收为字符串，转成对象，获取到linkcnt键对应的值，给当前事件源修改innerHTML。
//考虑多次请求，可能存在缓存状态，所以补充若是status为304也为请求成功。
datalist.onclick = function(e){
    if(e.target.className === 'like'){
        // e.target.innerText = e.target.innerText*1+1;
        // 获取当前微博id
        let currentLi = e.target.parentNode;
        let id = currentLi.dataset.id;
```

```
        let xhr = new XMLHttpRequest();
        let status = [200,304];
        xhr.onreadystatechange = function(){
            if(xhr.readyState === 4 && status.indexOf(xhr.status)>=0){
                let res = JSON.parse(xhr.responseText);
                e.target.innerHTML = res.likecnt;
            }
        }
        xhr.open('get','../api/weibo.php?id='+id,true);
        xhr.send();
    }
}
// (php)
//1.接收前端传过来的id，返回值为字符串
$id = isset($_GET['id']) ? $_GET['id'] : Null;
//2.打开json文件fopen()，读取json文件数据fread()。
$path = '';//相对路径
$file = fopen($path,"r");
$content = fread($file,filesize($path));
fclose($file);
//2.对得到的content内容转成json数组后，遍历里面每个item对象，若id键对应的值等于传入的$id,将该对象的
linkcnt++。
$contentArr = json.decode($content);//此处contentArr为对象数组
$res;
foreach($contentArr as $item){
    //此处item为一个对象
    if($item->id == $id){
        $item->linkcnt++;
        $res = $item;
    }
}
//4.将contentArr转成字符串，重新写入json文件
$file = fopen($path, 'w');
fwrite($file, json_encode($arr,JSON_UNESCAPED_UNICODE));
fclose($file);
//5.将步骤2声明变量res，将改变的item对象用res接收，转成字符串，echo给前端
echo json_encode($res,JSON_UNESCAPED_UNICODE);
```

补充：eval的使用

```
var json = '{"name":"lemon","age":18}'; //标准json字符串
var json = '{"name":"lemon",\'age\':18}';//不标准
eval('('+json+')');//也可以转成对象
eval('1+2');//3
eval('定义函数；执行函数') //函数可以在字符串中执行
```

讲解：ajax的来历及同源策略、同步异步

案例：用户名验证

```
username.onblur = function(){
    let _username = username.value;
    let status = [200,304];
```

```
    let xhr = new XMLHttpRequest();
    //xhr.onload意思为数据请求完成后，等同于xhr.readystate==4的状态
    xhr.onload = function(){
        if(status.includes(xhr.status)){
            let res = xhr.responseText;//fail,success
            if(res === 'fail'){
                username.nextElementSibling.innerHTML = `${_username}太受欢迎，你走吧`;
            }else{
                username.nextElementSibling.innerHTML = `恭喜你绿了`;
            }
        }
    }
    xhr.open('get','../api/checkname.php?username='+_username,true);
    xhr.send();
}
//php
//1.数组存放已经存在的用户名
$userlist = array('张三','李四','王尼玛','奥巴马','laoxie','lemon');
//2. 获取前端传来的用户名
$username = isset($_GET['username']) ? $_GET['username'] : null;
//3. 判断前端传来的用户名是否已存在数组内
$res = in_array($username, $userlist);
if($res){
    // 用户名已存在，注册失败
    echo "fail";
}else{
    echo "success";
}
```

案例：分页数据加载

```
//html页面：
var qty = 5;
// 1.发起ajax请求数据，拿到返回的数据，转成对象。
let status = [200,304]
let xhr = new XMLHttpRequest();
xhr.onload = function(){
    if(status.includes(xhr.status)){
        let res = JSON.parse(xhr.responseText);
        // 2.拿到对象的data键对应的值(为json字符串)，渲染页面
        datalist.innerHTML= res.data.map(function(item){
            return `<li>
            <h4>${item.name}</h4>
            <div>${item.content}</div>
            </li>`;
        }).join("");
        //3.得到总页码数pageNum，parseInt(数据总数res.total/每页数据qty).遍历生成页码
        var pageNum = parseInt(res.total/res.qty);
        page.innerHTML = "";
        for(var i=0;i<pageNum;i++){
            var span = document.createElement("span");
            span.innerHTML = i+1;
            //5.拿到返回的当前页码-1，把当前索引设置高亮
```

```
            var curidx = res.curpage -1;
            if(i == curidx){span.className = "active";}
            page.appendChild(span);
        }

    }
}
xhr.open('get','../api/lemon.php?qty='+qty+'&curpage=1',true);
xhr.send();
// 4.点击分页切换
page.onclick = function(e){
    if(e.target.tagName.toLowerCase() === 'span'){
        let curpage = e.target.innerText;
        xhr.open('get','../api/lemon.php?qty='+qty+'&curpage='+curpage,true);
        xhr.send();
    }
}
//php:
<?php
    $qty = isset($_GET["qty"])? $_GET["qty"] : 5;
    $curpage = isset($_GET["curpage"])? $_GET["curpage"] : 1;
    //1.拿到json数据,根据每页数量与当前页，裁切对应的数据
    $path = '../data/football.json';
    $file = fopen($path,'r');
    $content = fread($file,filesize($path));
    $data = json_decode($content);
    $curdate = array_slice($data,$qty*($curpage-1),$qty);
    //2.格式化数据，再返回给前端
    $res = array(
        "total" => count($data),
        "data" => $curdate,
        "qty" => $qty*1,
        "curpage" => $curpage*1,
    );
    echo json_encode($res,JSON_UNESCAPED_UNICODE);
?>
```

（三）ajax跨域解决方案

JSONP

- JSONP 是JSON with padding（填充式JSON 或参数式JSON）的简写。
- JSONP是一种可以绕过浏览器的安全限制，从不同的域请求数据的方法。
- JSONP请求不是ajax请求，是利用script标签能加载其他域名的js文件的原理，来实现跨域数据的请求
- 局限性：
    - 只能为get请求
    - 接口必须有回调函数的执行

演示：使用script标签其他js文件调用本地js的某个函数

```
//html页面:
<script type="text/javascript">
    function sum(n){
        console.log(n);
    }
</script>
<script type="text/javascript" src="../js/common.js"></script>

//common.js代码
sum(6666);

//此时html页面的控制台打印 6666。
```

演示：使用script标签其他php文件调用本地js的未知名方法，返回数据

```
//html页面:
<script type="text/javascript">
    function sum(data){
        console.log(data);
    }
    function sum2(data){
        alert(data);
    }
</script>
<script src="../api/lemon.php?callback=函数名"></script>

//php文件:
<?php
    $fn = $_GET["callback"];
    echo "$fn(数据)";
?>
```

案例：利用JSONP原理调用百度建议

```
msg.oninput = function(){
    let _msg = msg.value;
    clearTimeout(timer);
    timer = setTimeout(()=>{
    let script = document.createElement('script');
    script.src='https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/sujson=1&cb=getData&wd='+_msg;
    document.body.appendChild(script);
    },500);
}
window.getData = function(data){
    suggest.innerHTML = data.s.map(item=>{
        return `<li>${item}</li>`
        }).join("");
    }
})
//原理性代码:
//1.script的src中回调函数的传递
script.src='https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/sujson=1&cb=getData&wd='+_msg;
```

```
//2.声明全局函数
window.getData = function(data){处理数据}
```

CORS

CORS是一个W3C标准，全称是"跨域资源共享"（Cross-origin resource sharing），它允许浏览器向跨源服务器发出XMLHttpRequest请求，从而克服了AJAX只能同源使用的限制。

CORS需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，IE浏览器不能低于IE10。

局限性：服务器必须设置响应头，浏览器的兼容问题

（同样的发起ajax请求）

```
1.Access-Control-Allow-Origin
        header('Access-Control-Allow-Origin: * ');  //设置跨域请求必须要加上
该字段是必须的。需要在后端响应头添加词字段，值要么是一个*，表示接受任意域名的请求，要么指定一个域名
http://localhost。
2.Access-Control-Allow-Methods
3.Access-Control-Allow-Headers
    header('Access-Control-Allow-Methods:POST');
    header('Access-Control-Allow-Headers:x-requested-with,content-type');
```

案例：天气预报

```
//1.直接利用ajax请求得到数据
let xhr = new XMLHttpRequest();
let status = [200,304]
xhr.onload = function(){
    if(status.includes(xhr.status)){
        let res = JSON.parse(xhr.responseText)
        console.log(res.data);
        //拿到数据，可以渲染部分到页面上
        let title = document.createElement('h2');
        title.innerHTML = res.data.city + '天气预报';
        let tips = document.createElement('p');
        tips.innerHTML = res.data.ganmao;
        let ul = document.createElement('ul');
        ul.innerHTML = res.data.forecast.map(item=>{
            return `<li>
                <h4>${item.date}</h4>
                <p>${item.low} / ${item.high}</p>
                <p class="type">${item.type}</p>
            </li>`
        }).join('\n');
        // 清空内容
        weather.innerHTML = '';
        weather.appendChild(title);
        weather.appendChild(tips);
        weather.appendChild(ul);
    }
}
xhr.open('get','http://wthrcdn.etouch.cn/weather_mini?city=广州',true);
```

```
xhr.send();
//2.配合文本框实现，得到不同城市的天气预报
city.onblur = function(){
    let _city = city.value.trim();
    if(_city.length === 0){
        return;
    }
    xhr.open('get','http://wthrcdn.etouch.cn/weather_mini?city='+_city,true);
    xhr.send();
}
```

演示：百度地图

服务器代理

后端不存在跨域问题，所以可以利用后端间接获取其他网站的数据

```
ajax跨域请求之服务端代理（爬虫）
原理：获取页面所有内容，并利用正则匹配所需内容
file_get_contents($url) //获取网站内容
preg_match_all($reg,$str,$res)
preg_match($reg,$str,$res)   //$str代表被匹配的内容，$res为结果
$content = iconv(原字符编码,新字符编码,$content);//修改$content字符编码
```

案例：利用服务器代理获取外网IP

```
api地址：http://2018.ip138.com/ic.asp
//本地api代理
<?php
    $content = file_get_contents("http://2018.ip138.com/ic.asp");//拿到远程网站内容
    $content = iconv('gb2312','utf-8',$content);//修改网站内容字符编码与相应头一致
    preg_match_all('/\[([\d\.]+)\]/',$content,$res);//正则匹配
    echo $res[1][0];
?>
//前端访问本地api
```

案例：根据IP获取所在城市（ajax嵌套）

```
api地址：http://ip.taobao.com/service/getIpInfo.php?ip=$ip
//利用上一个案例请求拿到的ip，传入本地api。
let xhr_city = new XMLHttpRequest();
xhr_city.onload = function(){
    if(status.includes(xhr_city.status)){
        let city = xhr_city.responseText;
        //返回的是一串html结构加城市名，通过正则拿到城市名
        city = city.match(/[\u2E80-\u9FFF]+/)[0];
        console.log(city);
    }
}
xhr_city.open('get','../api/lemon.php?ip='+ip,true);
xhr_city.send(null);
//本地api
```

```
$ip = isset($_get['ip']) ? $_get['ip'] : "27.46.236.176";
$content = file_get_contents("http://ip.taobao.com/service/getIpInfo.php?ip=$ip");
$res = json_decode($content);
$data = $res->data;
$city = $data->city;
echo $city;
```

案例：根据城市获取天气预报（ajax嵌套）

- 所有的代码都写在一个function里面，数据混乱，容易出错。
- 维护困难

演示：post请求数据

Promise函数 （构造函数）

Promise是一个构造函数，所谓的Promise对象，就是通过new Promise()实例化得到的对象，用来传递异步操作的消息。它代表了某个未来才会知道结果的事件（通常是一个异步操作），并且这个事件提供统一的 API，可供进一步处理。

Promise 的三种状态

> Pending（未完成）可以理解为Promise对象实例new创建时候的初始状态 Resolved（成功） 可以理解为成功的状态 Rejected（失败） 可以理解为失败的状态

静态方法

Promise.all([p1,p2,p3...])

- 将多个Promise实例，包装成一个新的Promise实例
- 所有参数中的promise状态都为resolved时，新的promise状态才为resolved
- 只要p1、p2、p3..之中有一个被rejected，新的promise的状态就变成rejected

```
Promise.all([p1,p2,p3]).then(function(res){
        console.log(res);
})
```

Promise.race([p1,p2,p3...]) // 竞速，完成一个即可

原型方法

Promise.prototype.then(successFn[,failFn])

> Promise实例生成以后，可以用then方法分别指定Resolved状态和Rejected状态的回调函数。并根据Promise对象的状态来确定执行的操作: resolved时执行第一个函数successFn rejected时执行第二个函数failFn。

Promise.prototype.catch(failFn)

演示：定时器结束后再执行某些操作

```
var p = new Promise(function(resolve, reject){
        //做一些异步操作
        setTimeout(function(){
                console.log('执行完成');
                resolve('随便什么数据');
        }, 2000); }
);
p.then(function(data){
    //data为随便什么数据
})
```

演示：生成一个0-2之间的随机数，如果小于1，则等待一段时间后返回成功，否则返回失败：

```
function test(resolve, reject) {
    var timeOut = Math.random() * 2;
    console.log(timeOut);
    setTimeout(function () {
        if (timeOut < 1) {
            resolve('200 OK');
        }
        else {
            reject('timeout in ' + timeOut + ' seconds.');
        }
    }, 1000);
}
var p1 = new Promise(test);
var p2 = p1.then(function (result) {
    console.log('成功：' + result);
});
var p3 = p2.catch(function (reason) {
    console.log('失败：' + reason);
});
```

演示：利用promise完善ajax嵌套

> 有了Promise对象，就可以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

加载图片，获取图片信息（宽高）

```
var p = new Promise(function (resolve, reject) {
    var image = new Image();//生成img标签
    image.src = 'http://image.baidu.com/xxx.jpg;
    image.onload  = ()=>{
        resolve(image);
    }
    image.onerror = reject;
  });
//使用
p1.then((img)=>{
    document.body.appendChild(img);
    console.log(img.offsetWidth,img.offsetHeight);
})
```

try...catch

尝试执行代码，如果有错误则执行catch捕获错误(不报错)

```
try{
        console.log(666)
        //先尝试执行这里的代码
        show();
        //无报错，则忽略catch
        //如果报错，则执行catch，并传递错误信息
}catch(error){
        console.log(error)
}
console.log('end');
```

演示：xhr的兼容写法

```
var req = null;
try{
    req = new XMLHttpRequest();
}catch(err){
    // ie低版本浏览有多种异步请求的实现
    try{
        req = new ActiveXObject("Msxml2.XMLHTTP");
    }catch(err){
        try{
            req = new ActiveXObject("Microsoft.XMLHTTP");
        }catch(err){
            alert('你的浏览器太low了，这个世界不适合你');
        }
    }
}
```

#

#

==========

#

#

四、第四部分：面向对象+三层架构（界面层+业务逻辑层+数据访问层）

1.概念

将整个业务应用划分为：界面层（User Interface layer）、业务逻辑层（Business Logic Layer）、数据访问层（Data access layer）。

B层是通过后台语言（例如php）处理业务逻辑，必须建立在某个web服务（例如apache、iis）的基础上，才能通过浏览器进行访问。

D层是对数据库的操作，同样也必须建立在某个服务的基础上，才能保证别人能访问到数据库。

3. wampserver安装

- a:安装 Web 服务器Apache
- p:安装 PHP解析器
- m:安装MySQL数据库

> 对于初学者建议使用集成的服务器组件（如：WampServer），它已经包含了 PHP、Apache、Mysql 等服务,免去了开发人员将时间花费在繁琐的配置环境过程。

WampServer下载地址：http://www.wampserver.com/

若是缺乏某个dll，直接安装vc_redist.x64_2015.exe

4.iis+php+mysql

开启iis服务

控制面板\程序和功能\windows功能\internet信息服务所有选项都勾选上

验证：

控制面板/管理中心\iis manage（快捷方式拖拽至桌面）

开启服务-浏览器输入localhost:80-页面出现iis即成功

添加php-manage

安装 PHPManagerForIIS-1.2.0-x64.msi 及 php-7.0.13-nts-Win32-VC14-x64 .

验证：

上图第一个圆圈，后面的选项check phpinfo（）

运行出现正常网页效果即可。

mysql

安装mariadb

5.注意事项

iis服务与apache服务二选一，一方停止另一方才能开启。

mysql服务也是一样，不能同时开启多个。

如何关闭、开启服务：

计算机/管理/服务和应用程序/服务！！！

七、php与mysql交互

PHP 5 及以上版本建议使用以下方式连接 mysql数据库

MySQLi extension (“i” 意为 improved) 安装：Linux 和 Windows: 在 php5 mysql 包安装时 MySQLi 扩展多数情况下是自动安装

```
<?php
       $servername = "localhost";
       $username = "username";
       $password = "password";
       $dbname = 'user';

       // 创建连接
       $conn = new mysqli($servername, $username, $password, $dbname);

       // 检测连接
       if ($conn->connect_error) {
           die("连接失败: " . $conn->connect_error);
       }

       //查询前设置编码，防止输出乱码
       $conn->set_charset('utf8');

       echo "连接成功";
   ?>
```

1. 执行sql语句查询结果集

```
//编写sql语句
    $sql = 'select * from student';

    //获取查询结果集
    $result = $conn->query($sql);

    //使用查询结果集
    //得到数组
    $row = $result->fetch_all(MYSQLI_ASSOC);

    //释放查询结果集，避免资源浪费    只有查询才需要关闭查询结果集
    $result->close();

    //把结果输出到前台
    echo json_encode($row);

    // 关闭数据库，避免资源浪费
    $conn->close();
```

2. 插入数据 insert into 表名（字段名）values(要插入的记录)

单条数据

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql)) {
    echo "新记录插入成功";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
```

多条数据

```
$sql = "INSERT INTO `projects` (`name`, `url`, `description`) VALUES ";
    foreach ($projects as $item) {
        $sql .= "('$item',NULL, NULL),";
    }
    $sql = substr($sql,0,-1);

    if ($conn->query($sql)) {
        echo "新记录插入成功";
    } else {
        echo "Error: " . $sql . "<br>" . $conn->error;
    }
```

3. 查询数据（读取数据）

SELECT...FROM，得到查询结果集

- num_rows： 结果集中的数量，用于判断是否查询到结果
- fetch_all(MYSQLI_ASSOC) 得到所有结果
- fetch_assoc() 得到第一个结果
- fetch_row()

```
$sql = "SELECT id, firstname, lastname FROM MyGuests";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        // 输出每行数据
        while($row = $result->fetch_assoc()) {
            echo "<br> id: ". $row["id"]. " - Name: ". $row["firstname"]. " " .
$row["lastname"];
        }
    } else {
        echo "0 个结果";
    }
```

4. mysql 语句的返回值

```
返回布尔值
        insert
        update
        delete
返回查询结果集
        select语句
```

一、面向对象编程（自己总结）

（一）对象的创建

创建对象

1. 字面量 var obj = {}; 【一般用于单个对象】
2. 构造函数 new Object() 【一般用于单个对象】
3. 工厂函数 【产生的对象的构造函数都是Object，类不明确】

```
// 商品:
//   属性: 图片、标题、id、size、qty、价格、总价
//   行为:
//        增加商品数量
//        移除商品
//        清空购物车

function productGoods(imgurl,title,id,price){
    var goods = new Object();
    goods.imgurl = imgurl;
    goods.title = title;
    goods.id = id;
    goods.price = price;
    goods.init = function(){
        console.log(title+"初始化");
    }
    return goods;
}

var peiqi = productGoods("../images/1.jpg","小猪佩奇",1,50);
```

1. ```
   //工厂函数特点：生产出来的对象都是object类型
   ```
   ```

4. 自定义构造函数（类） new 自定义构造函数

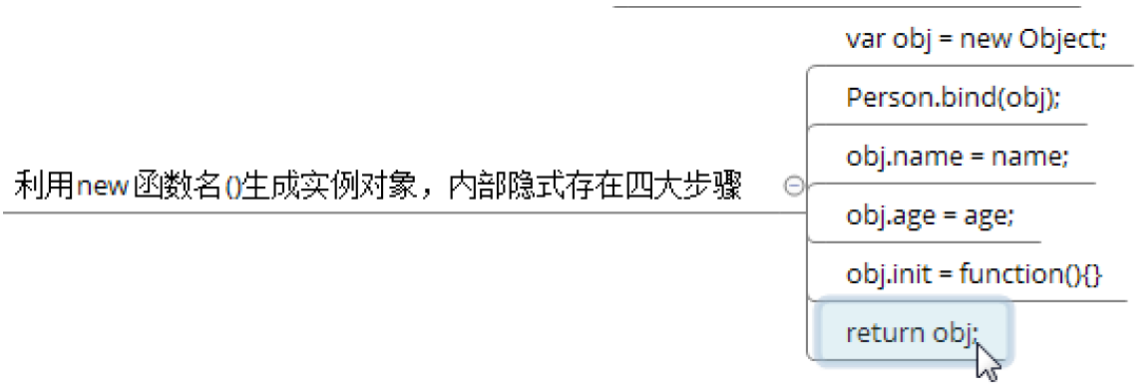
    1. 解决了工厂函数对象识别的问题
    2. 不成文的规定：为了区别书写构造函数与普通函数，构造函数首字母大写
    3. 区别：

        构造函数与普通函数没有本质区别，唯一的区别是执行方式【new 函数（） ===> 构造函数 ||| 函数（） ==> 普通函数，任何函数直接执行，没有return ， 返回值都是undefined

    4. new实例化对象的过程，相当与构造i函数内部执行了四大步骤：

        1. 创建一个新对象；
        2. 利用bind()方法改变this的指向
        3. obj.name = name
        4. return obj;

```
function yiwu(imgurl,title,id,price){
    // var peiqi = new Object();
    // yiwu.bind(peiqi);
    this.imgurl = imgurl;
    this.title = title;
    this.id = id;
    this.price = price;
    this.init = function(){
        console.log(this.title+"初始化");
    }
    // return peiqi;
}
```

    5. 

（二）对象的操作

1. 创建对象
2. 描述对象（键值对 “.” or "[]"）

    1. 属性：描述对象有什么
    2. 行为：描述对象能做什么

3. 指挥对象
    1. 行为执行的时候要加括号

构造函数：用new函数名（） 创建的对象，该函数就叫做构造函数

实例对象：用new函数名（）生成的对象称为实例对象

原型对象（prototype）

备注：

实例对象的**proto**指向原型对象；

原型对象的constructor(构造器)指向构造函数；

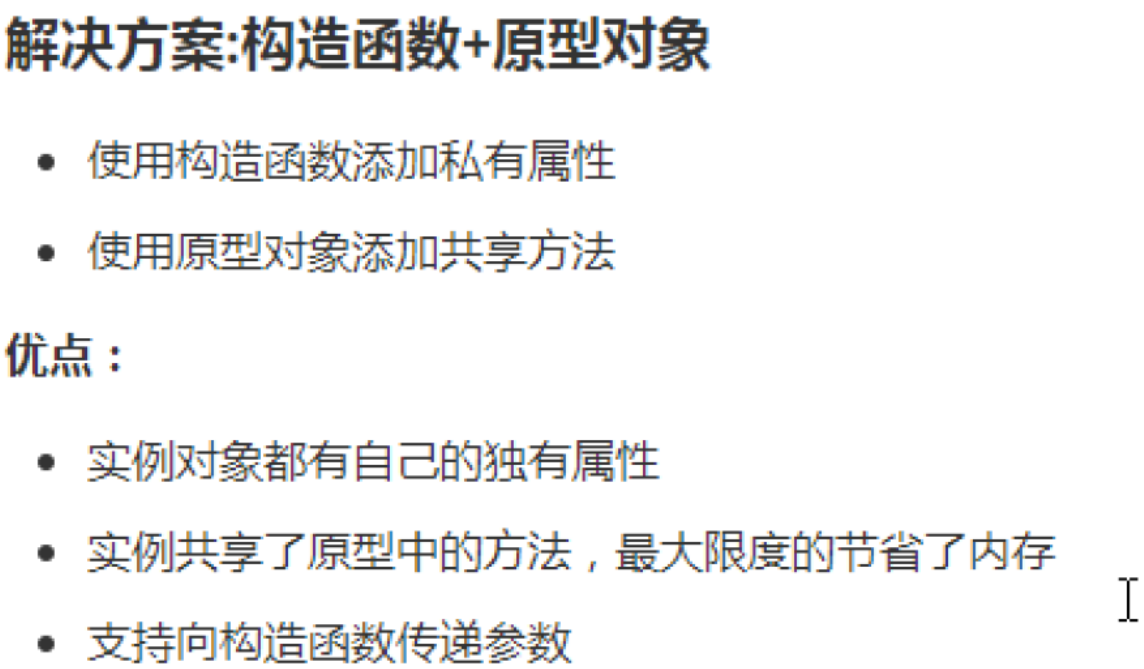
构造函数的prototype指向原型对象。

自定义构造函数+原型对象:

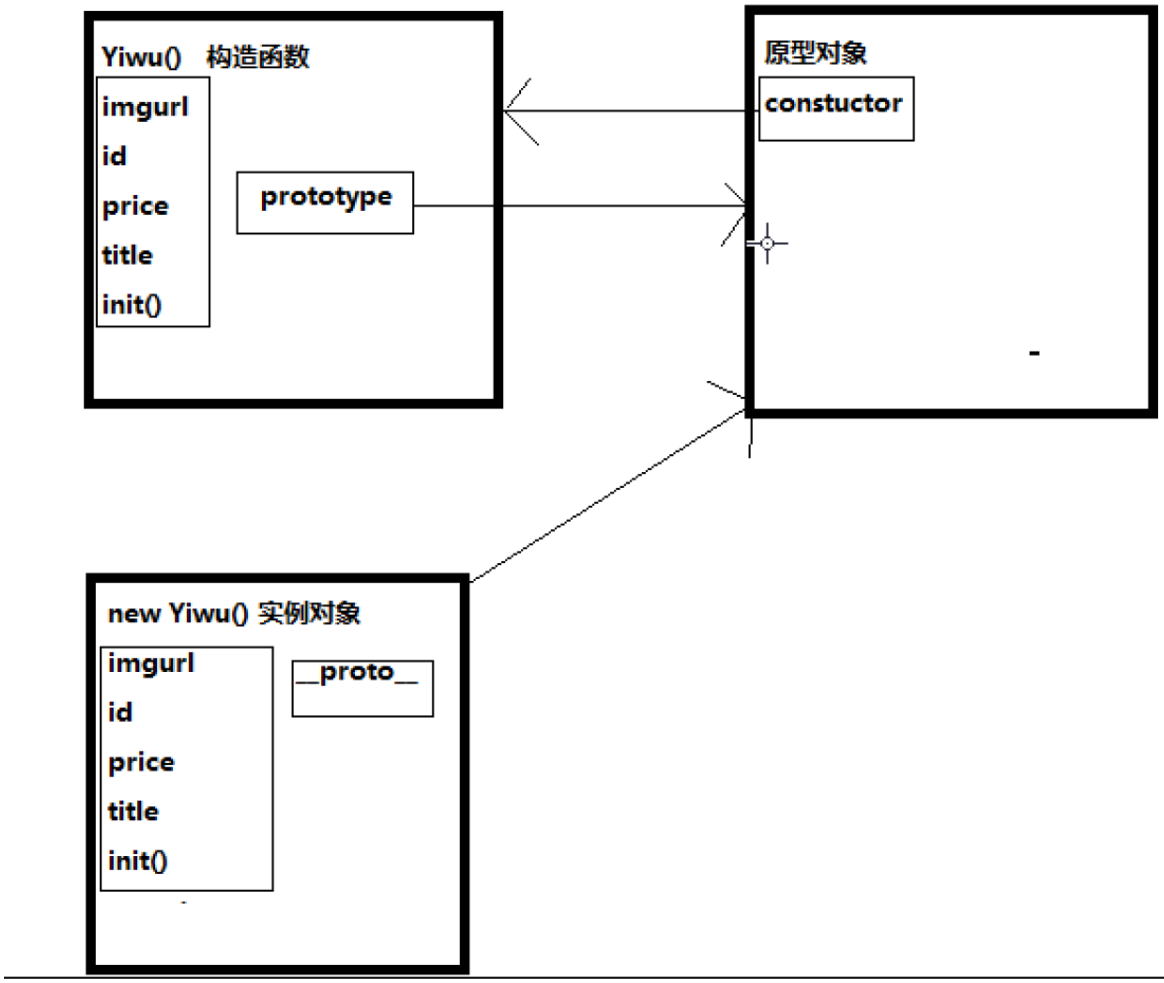
备注：

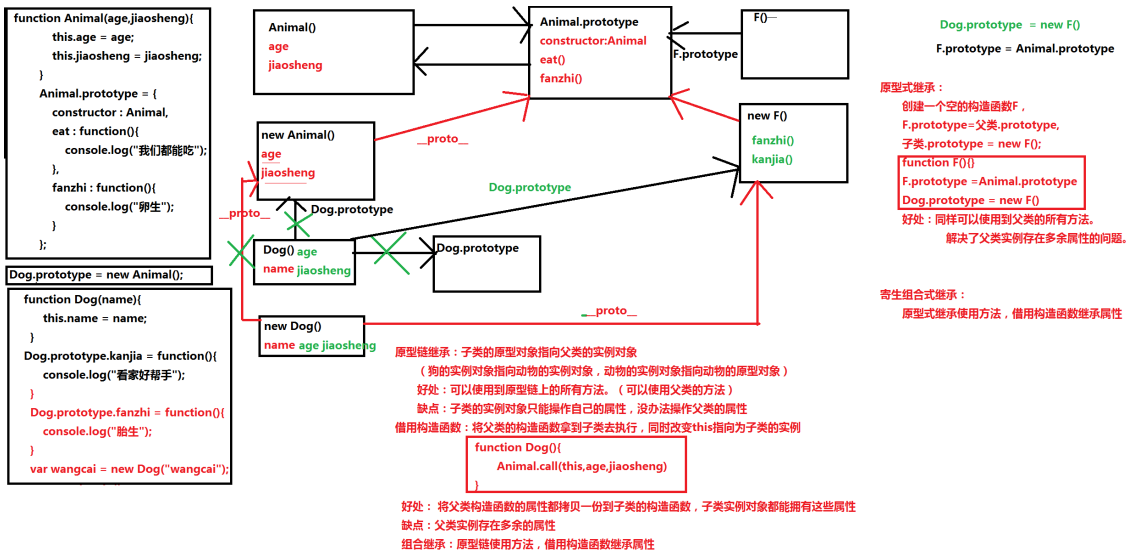
实例对象能使用原型对象的属性和方法，而不是拥有。实例对象拥有构造函数的属性和方法。

重点：创建对象时，若将所有的属性集方法都写在构造函数。会造成每个实例对象都拥有构造函数的所有的属性和方法，会造成内存的浪费。解决：私有属性写在构造函数；公共属性和方法写在原型对象。



原型链：





值属性的属性特性：

configurable:可配置性，控制着其描述的属性的修改，表示能否修改属性特性

enumerable:可每枚举性，表示能否修改通过for-in遍历得到的属性

writable：可写性，表示能否修改属性的值

value：数据属性，表示属性的值，默认值为undefined

【1.利用点或者[]操作对象的属性方式，默认以上的属性特征都为true（除了value为具体的值）】

操作属性特性及属性的的方法

object.defineProperty(obj,property,descriptor) 给对象的某个属性设置属性特性

```
//  2.操作属性及属性特性的方法
//  Object.defineProperty(obj, property, descriptor) 给对象的某个属性设置属性特性
```

二、面向对象

利用对象进行编程的一种思想。object-oriented programming，简称oop。

（一）面向过程及面向对象的区别

举例:五子棋游戏

```
//面向过程：
1、开始游戏，
2、黑子先走，
3、绘制画面，
4、判断输赢，
5、轮到白子，
6、绘制画面，
7、判断输赢，
8、返回步骤2，
9、输出最后结果。
//面向对象
1、黑白双方，这两方的行为是一模一样的．  位置、种类{black:[],white:[]}
2、棋盘系统，负责绘制画面。盘、黑白子、[创建span，添加不同类名，位置。]
3、规则系统，负责判定诸如犯规、输赢等。判断是不是在同一条线上
//总结:面向对象是按照功能划分问题，保证了可扩展性。例如添加悔棋功能，如果是面向过程，那么从输入到判断到显示这一连串的步骤都要改动，甚至步骤之间的循序都要进行大规模调整。而面向对象，只需要让棋盘系统回溯到上一步即可，无需纠结1、3功能。
```

演示：创建并描述对象

- 描述一个人
- 描述购物车

（二）如何创建对象

1.字面量 【用于创建单个对象】

var student = {id:10,name:'小明',age:18}

2.通过new关键字实例化对象 【用于创建单个对象】

```
var student = new Object()
student.id = 10;
student.name = '王铁锤';
student.age = 18;
//缺点：使用同一个接口创建很多对象，会产生大量的重复代码
```

工厂模式（可以创建多个对个对象、产生的对象的构造函数都是Object，类不明确）

在ECMAScript中是无法创建类的，开发人员就发明了一种函数，用函数来封装特定接口创建对象的细节。

```
function createPerson(name, age, job) {
    var o = new Object();
    o.name = name;
    o.age = age;
    o.job = job;
    sayName = function () {
        alert(this.name);
    };
    return o;
}
var person1 = createPerson('zxj', 23, "Software Engineer");
var person2 = createPerson('sdf', 25, "Software Engineer");

//不足：在示例中我们可以看到，工厂模式虽然解决了创建多个相似对象的问题，但没有解决对象识别的问题（在示例中，得到的都是o对象，对象的类型都是Object）。
```

3.自定义构造函数（类的概念） 【常用】

ECMAScript中的构造函数可以用来创建特定类型的对象。像Object和Array的原生的构造函数，在运行时会自动出现在执行环境中。此外，也可以创建自定义的构造函数，从而定义自定义对象类型的属性和方法。

演示：自定义构造函数相对于工厂函数的好处

```
function Student(name, age){
    this.name = name;
    this.age = age;
    this.sayName = function(){alert(this.name);};
}
var s1 = new Student("王铁锤", 18);
//可以通过控制台查看一下person1与s1的区别，更好理解。
```

构造函数与普通函数的区别

> 唯一区别：调用方式不同

- 任何函数，只要通过new操作符来调用，它就可以作为构造函数；
- 而任何构造函数，如果不通过new 操作符来调用，那它跟普通函数无区别。

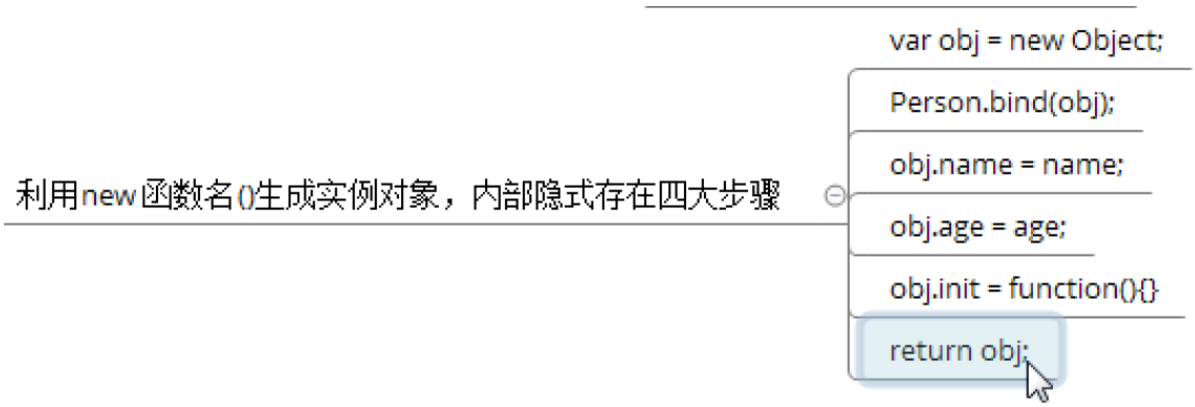
> 不成文的约定：构造函数名首字母大写

演示：构造函数与普通函数执行方式的区别

```
function Dog(){
        this.name = '哈巴';
        this.color = '白色';
        this.jiao = function(){
        console.log("我的名字叫"+this.name);
    }
}
// 普通函数执行方式
// Dog(); => 没有值return出来，所以输出结果为undefined
// 构造函数的调用方式
// new Dog(); => 输出结果为对象
```

*** 调用自定义构造函数实际上会经历以下4个步骤：**

1. 创建一个新对象(在内部隐式调用了new Object())；
2. 将构造函数的作用域赋给新对象（把this绑定到实例对象）；
3. 执行构造函数中的代码（为这个新对象添加属性）；
4. 返回新对象。




```
// 通过new调用函数==>构造函数
// 1. var obj = {}
// 2. Dog.bind(obj)
// 3. 执行构造函数中的代码：this为obj
// 4. return obj;
```

（三）this 的指向问题

函数中的this作为JS的关键字，有特殊的含义，代表了当前对象，而当前对象是谁，由函数执行时所处的环境来决定。

用new关键字执行：this指向生成的实例对象 普通函数执行：this指向调用函数的对象

注：在js的整理笔记中有详细的介绍

（四）对象的组成部分：（构造函数、实例对象、原型对象）

构造函数

- this指向实例对象

实例对象

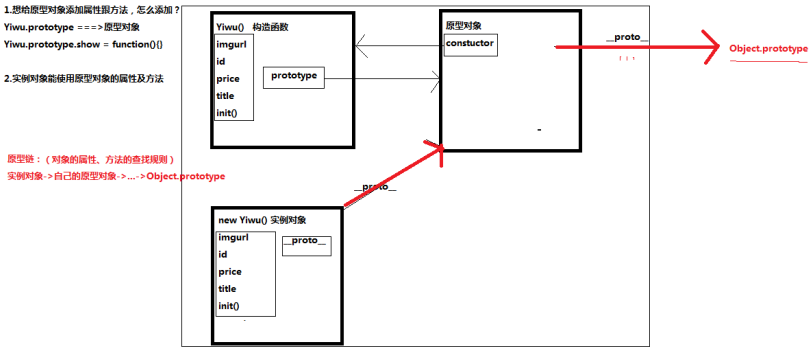
1. 用new关键字生成的对象称为实例，实例会复制构造函数内所有的属性和方法。
2. 可以实例对象获取原型对象：

```
__proto__  //Frefox,Chrome等浏览器可以通过私有属性
Object.getPrototypeOf(实例);  //ES5方式去获取
```

原型对象

- 我们创建的每个函数都有一个prototype属性，指向原型对象。
- 原型对象默认包含一个constructor属性，指向构造函数。
- **任何写在原型对象中的属性和方法，都可以让所有实例对象共享。**

演示：画图演示三者的关系



三者的关系

1. 每个构造函数都有一个原型对象（prototype），
2. 原型对象都包含一个指向构造函数的指针（constructor），
3. 而所有实例都通过`__proto__`指向同一个原型对象。（包含一个指向原型对象的内部指针（[[prototype]]））

判断原型和实例的关系（返回布尔值）

constructor: 得到构造函数的引用，一般用于判断该实例是否由某一构造函数生成

```
实例.constructor == Student //true
```

instanceof: 检测某个对象是不是某一构造函数的实例，适用于原型链中的所有实例

```
实例 instanceof Student //true
实例 instanceof Object //true
```

isPrototypeOf: 判断当前对象是否为实例的原型对象

```
原型对象.isPrototypeOf(实例) //true
```

（五）实际应用

构造函数方法很好用，但是单独使用存在一个浪费内存的问题（所有的实例会复制所有构造函数中的属性/方法）。这样既不环保，也缺乏效率。

解决方案:构造函数+原型对象

- 使用构造函数添加私有属性
- 使用原型对象添加共享方法

优点:

- 实例对象都有自己的独有属性
- 实例共享了原型中的方法，最大限度的节省了内存
- 支持向构造函数传递参数

演示：使用原型对象添加共享方法

```
function Person(name,age,gender){
    // 属性
    this.name = name;
    this.age = age;
    this.gender = gender;
    // 方法
    // this.say = function(){console.log('say');}
    // this.singing = function(){console.log('singing');}
}
 // 把方法提取到原型对象
 Person.prototype.say = function(){
    console.log('say');
 }
 Person.prototype.singing = function(){
    console.log('singing');
 }
 var lx = new Person('laoxie',18,'male');//2M
 var lm = new Person('lemon',31,'femal');//5M
```

案例：弹幕效果

```
//1.生成页面元素对象
//属性有：整个弹幕区域、消息框、按钮
//方法有：
//       -初始化：获取元素，点击按钮，执行发送方法
//       -发送方法，获取消息框内容。实例化弹幕对象
```

```
let page = {
    ele:'#barrage',
    msg:'#msg',
        button:'#msg+button',
    init(){
        this.ele = document.querySelector(this.ele);
        this.msg = document.querySelector(this.msg);
        this.btn = this.msg.nextElementSibling;
        this.btn.onclick = ()=>{
            this.send();
        }
    },
    send(){
        let msg = this.msg.value;
        new Barrage(msg);
    }
}
```

2.自定义弹幕对象的构造函数

(1) 属性：内容、颜色、速度、字体大小、位置

(2) 方法：

- 初始化init()：创建弹幕对象this.ele，添加内容类名及样式，添加到元素内
- 运动move()：从右往左
- 移除remove():移除弹幕

```
function Barrage(msg){
    this.color = randomColor();
    this.speed = randomNumber(-20,-5);
    this.size = randomNumber(12,48);
    this.position = randomNumber(10,page.ele.clientHeight-this.size-10);
    this.init(msg);
}
Barrage.prototype.init = function(msg){
    // 创建弹幕元素
    this.ele = document.createElement('span');
    this.ele.innerText = msg;
    this.ele.className = 'bar-item';
    // 定义样式
    this.ele.style.color = this.color;
    this.ele.style.fontSize = this.size + 'px';
    this.ele.style.top = this.position + 'px';
    // 写入页面
    page.ele.appendChild(this.ele);
    // 移动
    this.move();
}
Barrage.prototype.move = function(){
    this.timer = setInterval(()=>{
        let left = this.ele.offsetLeft;
        left += this.speed;
        if(left <= -this.ele.offsetWidth){
            clearInterval(this.timer)
            this.remove();
        }
        this.ele.style.left = left + 'px';
```

```
    },30);
}
Barrage.prototype.remove = function(){
    this.ele.parentNode.removeChild(this.ele);
}
// 页面元素对象的init方法开始执行
page.init();
```

案例：烟花效果

```
* 1.页面 page （字面量：一个）
        * 属性： 按钮、显示区域
        * 方法： 初始化（获取元素，给this.ele绑定点击事件,获取光标位置，实例化烟花对象）
let page = {
    ele:'body',
    btn:'#auto',
    init(){
        this.ele = document.querySelector(this.ele);
        this.btn = document.querySelector(this.btn);
        document.onclick = function(e){console.log(666)
            new Firework(e.pageX,e.pageY);
        }
    }
}
* 2.烟花 firework
        * 属性：位置、爆炸数量、爆炸半径
        * 方法：
                (1) 初始化：生成span，添加类名，及当前的水平坐标，添加到页面中。执行移动方法
                (2) 移动： 利用动画移动到光标top位置，同时将自身高度变小。执行爆炸方法，同时移除自身。
                (3) 爆炸：利用数量计算角度，将圆心位置，半径及角度都传给火花实例对象
                (4) 移除
function Firework(x,y){
    this.left = x;
    this.top = y;
    // 爆炸数量
    this.qty = randomNumber(12,36);
    // 爆炸半径
    this.r = randomNumber(50,200);
    this.init();
}
Firework.prototype.init = function(){
    this.ele = document.createElement('span');
    this.ele.className = 'fire';
    this.ele.style.left = this.left + 'px';
    page.ele.appendChild(this.ele);
    this.move();
}
Firework.prototype.move = function(){
    animate(this.ele,{top:this.top,height:this.ele.clientWidth},()=>{
        this.boom();
        this.remove();
    })
}
```

```
Firework.prototype.boom = function(){
    for(var i=0;i<this.qty;i++){
        // 计算角度
        let deg = 360/this.qty*i;
        new Spark(this.left,this.top,this.r,deg);

    }
}
Firework.prototype.remove = function(){
    this.ele.parentNode.removeChild(this.ele);
}
* 3.爆炸烟火 spark
        * 属性：随机色、半径、弧度、圆心位置
        * 方法：
                初始化：创建this.ele火花span、添加类名，设置初始化位置，写入页面
                火花移动:设置最终位置，利用动画移动到指定left、top值，及改变透明度。动画完成后移除
                移除火花
function Spark(x,y,r,deg){
    this.color = randomColor();
    this.r = r;
    this.rad = Math.PI*deg/180;
    this.init(x,y);
}
Spark.prototype.init = function(x,y){
    this.ele = document.createElement('span');
    this.ele.className = 'spark';
    // 设置颜色
    this.ele.style.backgroundColor = this.color;
    // 设置初始位置
    this.ele.style.left = x + 'px';
    this.ele.style.top = y + 'px';
    // 写入页面
    page.ele.appendChild(this.ele);
    this.move(x,y)
}
Spark.prototype.move = function(x,y){
    var a = this.r * Math.cos(this.rad);
    var b = this.r * Math.sin(this.rad);
    var left = parseInt(x + b);
    var top = parseInt(y - a);
    animate(this.ele,{left,top,opacity:0.3},()=>{
        this.remove();
    })
}
Spark.prototype.remove = function(){
        this.ele.parentNode.removeChild(this.ele);
}
// 操作对象
page.init();
```

（六）属性特性:ES5对象扩展(了解)

值属性的属性特性

- configurable
  - 可配置性，控制着其描述的属性的修改，表示能否修改属性特性
- enumerable
  - 可枚举性，表示能否通过for-in遍历得到属性
- writable
  - 可写性，表示能否修改属性的值
- value
  - 数据属性，表示属性的值。默认值为undefined

与属性特性相关的方法

```
设置属性特性：

Object.defineProperty(obj, property, descriptor) 给对象的某个属性设置属性特性
Object.defineProperties(object, descriptors) 给对象的所有属性设置属性特性

获取属性特性：

Object.getOwnPropertyDescriptor(object, propertyname)

获取对象的所有属性：

Object.keys(object) 只获取到可枚举的属性
Object.getOwnPropertyNames(object) 获取所有属性的名称（包含不能枚举的属性）
```

演示：添加属性的相关方法（同时设置属性特性）

```
var obj = {
    a:1,
    b:"lemon"
}
//1.给对象添加一个属性，同时设置属性特性。此时其他的属性特性不设置默认都为false。
//即configurable为false，不可以修改属性特性。
//即enumerable为false，不可以枚举
//即writable为false，不可写，不可以修改属性值
Object.defineProperty(obj,"c",{value:"laoxie"});  //以上属性特性均可改成true，实现相应的功能。

// 2.同时修改多个属性特性
Object.defineProperties(obj, {
    b:{value:'css4',enumerable:false},
    c:{writable:true}  //报错，因为上面定义该属性的configurable为false，意思是不可以修改属性特性
})

//建议：用传统方式添加属性，利用defineProperty修改属性特性（configurable为true的前提下）
```

（七）对象属性的遍历与判断

for...in

遍历对象中的所有可枚举属性, 无论该属性存在于实例中还是原型中

in

```
if(name in s1){
        //只要通过对象能够访问到属性就返回true，无论该属性存在于实例中还是原型中
}
```

对象.hasOwnProperty(属性)

- 检测一个属性是存在于对象本身中
    - 返回true，说明属性存在对象中
    - 返回false，说明属性不存在或在原型中

> 检测一个属性是否存在于原型中：!obj.hasOwnProperty(name) && (name in obj)

重置原型对象

重置原型对象，可以一次性给原型对象添加多个方法，但切断了与原来原型对象的联系

```
function Popover(){}
Popover.prototype = {
    show:function(){},
    hide:function(){}
}
//- 注意覆盖问题（系统的原型对象不建议重写）
//- 注意识别问题（不可枚举性）
Object.defineProperty(Popover.prototype,"constructor",{value:"Popover",configurable:true});
```

内置构造函数的原型对象

使用内置原型可以给已有构造函数添加方法

- 数组/字符串/数字等方法调用原理
- 扩展内置方法

演示：对象的内置函数及扩展内置函数

```
if(!Array.prototype.norepeat){
    Array.prototype.norepeat = function(){
        return Array.from(new Set(this)); // this：指向实例arr
        }
}
```

二、闭包

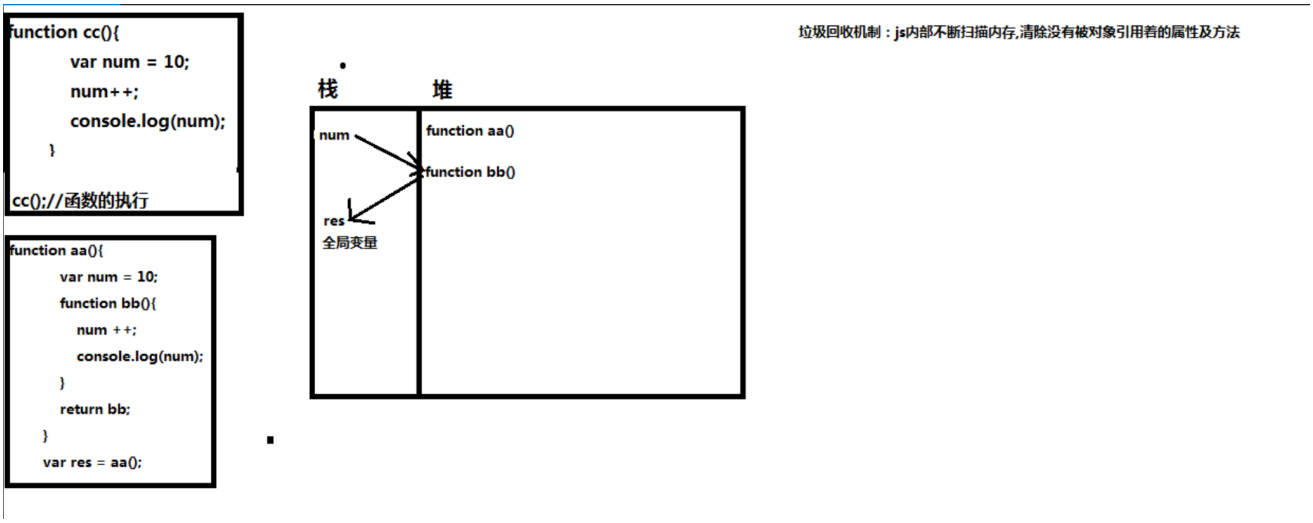
1.闭包的定义：

外层函数嵌套内函数，在外函数讲内函数返回出去，内部函数可以引用外部函数的参数和变量，参数和变量不会被垃圾回收机制所回收.

2.垃圾返回机制：

当一个函数执行完毕时，并清除没有被对象引用的属性及方法

当一个函数执行完毕时，会清除内部的没有被引用的局部变量。全局变量不会被回收（尽量避免定义全局变量）



3.闭包的用法：

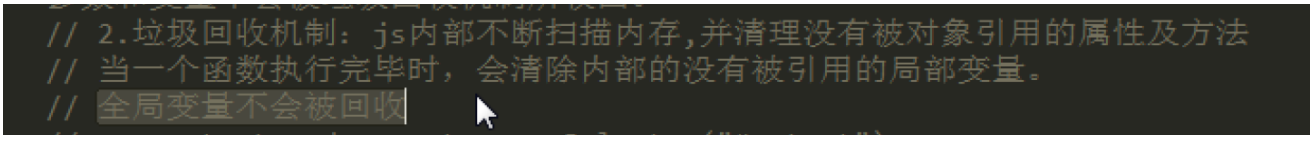
在函数内部想使用函数内部的属性和方法【例如点击按钮获取对应的索引值】

异步应用

参数和变量不会被垃圾回收机制所回收。

```
function aa(){
    var num = 10;
    function bb(){
        num++;
        console.log(num);
    }
    return bb;
}

//bb();  在这里无法访问内部的函数
var res = aa();   //此时res 接收到的是aa 的返回值 bb;
```



4.闭包的好处：

1. 可以让一个变量长期驻扎在内存当中不被释放
2. 避免全局变量的污染, 和全局变量不同, 闭包中的变量无法被外部使用
3. 私有成员的存在, 无法被外部调用, 只可以自己内部使用

> 结论：

- 闭包是指有权访问另一函数作用域中的变量的函数
- 闭包，可以访问函数内部的局部变量，并让其长期驻留内存
- 由于闭包会携带包含它的作用域(运行环境)，因此会比其他函数占用更多内存，过度使用闭包可能会造成性能问题。

案例：

- 点击按钮打印当前索引值
- tab标签切换

三、原型链【实例与object原型对象之间的链条成为原型链】

（一）原型模式的访问机制（原型搜索机制）：

1. 读取实例对象的属性时，先从实例对象本身开始搜索。如果在实例中找到了这个属性，则返回该属性的值；
2. 如果没有找到，则继续搜索实例的原型对象，如果在原型对象中找到了这个属性，则返回该属性的值
3. 如果还是没找到，则向原型对象的原型对象查找，依此类推，直到Object的原型对象（最顶层对象）；
4. 如果再Object的原型对象中还搜索不到，则抛出错误；

（二）重置原型对象

可以一次性给原型对象添加多个方法，但切断了与原来原型对象的联系

```
function Popover(){}
Popover.prototype = {
    show:function(){},
    hide:function(){}
}

注意覆盖问题
注意识别问题
```

（三）内置原型对象

使用内置原型可以给已有构造函数添加方法

- 数组/字符串/数字等方法调用原理
- 扩展内置方法

对象属性的遍历与判断

1.for...in: 遍历对象中所有枚举属性，无论该属性存在于实例中还是原型中

2.in:只要通过对象能够访问到的属性就返回true，无论该属性存在于实例中还是原型中

```
if (name in s1) {

}
```

1. 对象.hasOwnProperty(属性)：检测一个属性是存在于对象本身中。
    - 返回true，说明属性存在对象中
    - 返回false，说明属性不存在或在原型中

    检测一个属性是否存在于原型中：`!obj.hasOwnProperty(name) && (name in obj)`

三、继承

继承是面向对象中一个非常重要的特征。指的是：子类继承父类的属性和方法。

1.继承的好处：

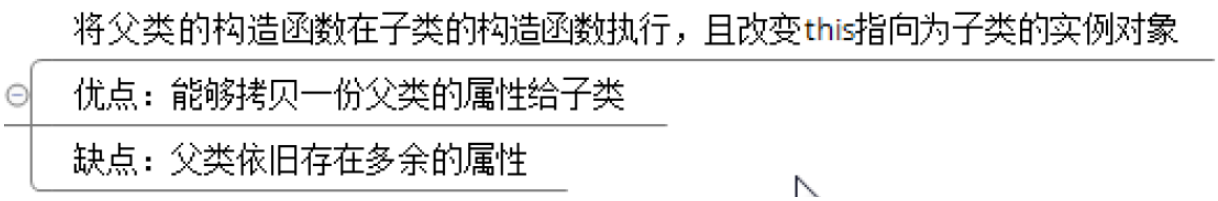
1. 子类拥有父类所有的属性和方法（代码复用）；
2. 子类可以扩展自己的属性和方法（更灵活）；
3. 子类可以重写父类的方法

2.继承方式

（一）原型链继承

- 核心：拿父类实例来充当子类原型对象，能使用原型链上的属性方法
- 缺点：
    - 无法继承构造函数中的属性
    - 创建子类实例时，无法向父类构造函数传参
    - 原型对象中存在多余的属性

（二）借用构造函数



核心：借父类的构造函数来增强子类实例，相当于把父类的实例属性复制一份给子类实例属性。

call 用法 : 父类构造函数.call(子类实例,参数1,参数2,参数3...)

apply 用法 ： 父类构造函数.apply(子类实例,[参数1,参数2,参数3...])

【call与apply的唯一区别：传参方式不同，call多个参数，apply只有两个参数，第二个参数为数组 】

缺点：

1.无法实现函数复用

2.函数太多就影响性能，占用更多内存

（三） 组合继承 【原型链使用方法，借用构造函数拷贝属性】

由于以上继承方法的缺点，实际开发中不可能单纯的只使用一种继承方法，而是利用它们的优点，规避它们的缺点，所以就有了组合继承法 。

继承属性：借用构造函数

继承方法：原型链继承

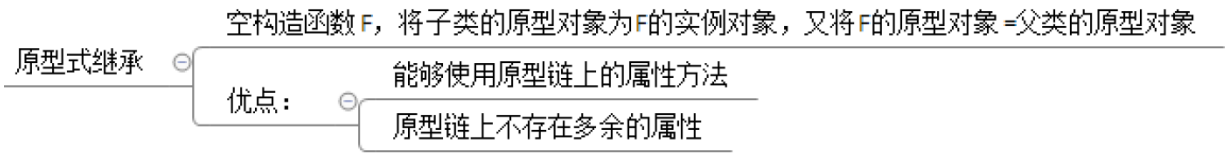
组合继承：最常用的继承模式。

缺点（原型链继承法的缺点）：

在原型对象中生成多余的属性

多次执行父类构造函数

（四）原型式继承



核心：先创建了一个临时性的构造函数，然后将传入的对象作为这个构造函数的原型，最后返回了这个临时类型的一个新实例 【解决原型链继承法的缺点：生成多余的属性 】

```
function object(o){
        function F(){}
        F.prototype = o;
        return new F();
    }
```

- ES5版本的原型式继承：Object.create()

（五）寄生组合继承法 【使用原型式使用方法+借用构造函数拷贝属性】

完美的继承方法

核心：

继承属性【借用构造函数】

继承方法【原型式继承】

（六）ES6中的继承 【class】

class

constuctor(参数){构造函数}

原型对象的方法：

init（）{}

extends继承（原型式继承）

class Dog extends Animal

super(属性) 拷贝父类构造函数的属性

constuctor(){super(属性)}

static静态方法

【静态方法必须通过构造函数去调用，或者类去调用】

Class定义类

ES6提供了更接近传统语言的写法，引入了Class（类）这个概念，作为对象的模板。通过class关键字，可以定义类

```
//定义类
class Person {
    constructor(name,age) {
        this.name = name;
        this.age = age;
    }
    getInfo() {
         return `我叫${this.name},今年${this.age}岁`;;
    }
}

写在类里面的方法实际是给Person.prototype添加方法
constructor方法是类的默认方法，通过new命令生成对象实例时，自动调用该方法。如果没有constructor方法，则得使用默认的constractor方法
```

extends继承

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
}

class Man extends Person {
    constructor(name, age, gender) {
        //this.gender = gender; // 报错
        super(name, age);
        this.gender = gender; // 正确
    }
}
```

- 子类继承了父类，在子类构造函数中必须调用super方法。
- 子类的constructor方法没有调用super之前，不能使用this关键字，否则报错，而放在super方法之后就是正确的。

（七）静态方法 【如果在一个方法前，加上static 关键字，否则报错，而放在super方法之后就是正确的。】

```
class Person {
    constructor(){
        this.name = 'laoxie',
        this.age = 18;
    }
    static getInfo(){
        return this.name
    }
    say(){
        console.log(`Hello everyone, my name is ${this.name}, I'm ${this.age} years old`)
    }
}
class Man extends Person {}
```

- 静态方法方法不会被实例继承，而是直接通过类来调用`Person.getInfo()`
- 父类的静态方法，可以被子类继承`Man.getInfo()`

案例：1.扩展原生js的方法

兼容性字符串的trim方法

获取ASCII码的方法

反编译ascii码的方法

版本管理工具

svn

- RCS（Revision Control System）
  即程序改版控制系统，主要功能是用来管理文件的版本，可以节省空间和时间。这样就不需要在每个程序开发到某一个阶段就将数据拷贝到其他的地方备份起来了。
- CVS（Concurrent Version System）
  现代管理里工具始祖
- SVN（Subversion）
  集中式管理工具的集大成者
- GIT
  分布式版本管理工具（后面讲解）

SVN是Subversion的简称，是一个**集中式**代码管理版本控制系统，相较于RCS、CVS，它采用了分支管理系统，它的设计目标就是取代CVS。说得简单一点SVN就是用于多个人共同开发同一个项目，共用资源的目的。

详细信息在单独的SVN文档里

插件

（一）弹窗插件

```
function Popover(options){
        //1.设置默认值，与传进来的对下那个进行合并
        var defaults = {
                width:600,
                height:'auto',//可以是数值
                position:'center',//可以传进来对象{x,y}
                title:'弹窗标题',
                content:'',
                draggable:true,
                overlay:0.3 //数字：有遮罩层，false：无遮罩层
        }
    var opt = Object.assign({},defaults,options);
    this.position = opt.position;
    this.init(opt);
}
Popover.prototype = {
        init(opt){
                //2.初始化方法：
                // （1）创建弹窗this.ele，设置宽高
                this.ele = document.createElement('div');
                this.ele.className = 'popover';
        this.ele.style.width = opt.width + 'px';
        if(typeof opt.height === 'number'){
                this.ele.style.height = opt.height + 'px';
        }
                // （2）标题和内容（设置类名title、content）
        var title = document.createElement('div');
        title.className = 'title';
        title.innerHTML = opt.title;
        this.ele.appendChild(title);
        var content = document.createElement('div');
        content.className = 'content';
        content.innerHTML = opt.content;
        this.ele.appendChild(content);
        // (3)遮罩层(判断overlay是数字则为透明度，为false则不设置遮罩层)（类名、透明度）
        if(opt.overlay !== false){
                this.bg = document.createElement('div');
                this.bg.className = 'overlay';
                this.bg.style.opacity = opt.overlay;
                document.body.appendChild(this.bg);
        }
        //  （4）删除按钮（类名、内容）
        var btnClose = document.createElement('span');
        btnClose.className = 'btn-close';
        btnClose.innerHTML = '&times;';
        this.ele.appendChild(btnClose);
        //  （5）弹窗写入页面，并显示弹窗
```

```
        document.body.appendChild(this.ele);
        this.show();
        //  (6) 设置关闭事件
        btnClose.onclick = function(){
                this.close();
        }.bind(this);
        // (7) 拖拽
        if(opt.draggable){
            this.drag();
        }
        },
        //3.1显示：设置成块，同时调用定位方法
        show(){
                this.ele.style.display = 'block';
                if(this.bg){
                        this.bg.style.display = 'block';
                }
                this.setPosition();
        },
        //3.2隐藏：
        close(){
                this.ele.style.display = 'none';
                if(this.bg){
                        this.bg.style.display = 'none';
                }
        },
        //4.定位：判断是否传参
        // (1) 若没有参数，即x值为undefined，则判断this.postion的值为center还是对象，设置x、y值
        // (2) 给元素设置样式
        setPosition(x,y){
                if(x===undefined){
                        // 如果元素没有添加(并显示)到页面，获取不到宽高
                        if(this.position === 'center'){
                                x = (window.innerWidth - this.ele.offsetWidth)/2;
                                y = (window.innerHeight - this.ele.offsetHeight)/2;
                        }else if(typeof this.position === 'object'){
                                x = this.position.x;
                                y = this.position.y;
                        }
                }
                this.ele.style.left = x + 'px';
                this.ele.style.top = y + 'px';
        },
        //5.拖拽
        drag(){
                var self = this;
                var pop = self.ele;
                pop.onmousedown = e=>{
                        var ox = e.clientX - pop.offsetLeft;
                        var oy = e.clientY - pop.offsetTop;
                        // 只能在标题位置拖拽
                        if(oy>pop.children[0].offsetHeight){
                                return;
```

```
                        }
                        document.onmousemove = function(evt){
                                var x = evt.clientX - ox;
                                var y = evt.clientY - oy;
                                self.setPosition(x,y);
                                evt.preventDefault();
                        }
                }
                document.onmouseup = function(){
                        document.onmousemove = null;
                }
        }
```

弹窗继承

```
//确认对话框
function Confirm(options){
        var defaults = {
                width:300,
                title:false,
                content:'你确定这个操作吗',
                overlay:false,
                confirm:function(){},
                cancel:function(){}
        }
    var opt = Object.assign({},defaults,options);
    //1.借用构造函数，拿到弹窗对象属性this.ele及this.position
    Popover.call(this,opt);
}
// 2.继承Popover的方法
Confirm.prototype = Object.create(Popover.prototype);
//3.添加/重置方法
Confirm.prototype.init = function(opt){
        //3.1把你需要的部分复制下来
        //3.2添加确认取消按钮
        this.confirmBtn = document.createElement('button');
        this.confirmBtn.className = 'btn-confirm';
        this.confirmBtn.innerText = '确认';

        this.cancelBtn = document.createElement('button');
        this.cancelBtn.className = 'btn-cancel';
        this.cancelBtn.innerText = '取消';
        this.ele.appendChild(this.confirmBtn);
        this.ele.appendChild(this.cancelBtn);

        // 3.3点击确认取消按钮，执行方法
        this.confirmBtn.onclick = ()=>{
                opt.confirm();
                this.close();
        }
        this.cancelBtn.onclick = ()=>{
                opt.cancel();
```

```
                this.close();
        }
```

（二）时间格式化

```
if(!Date.prototype.format){
    Date.prototype.format = function(fmt){
        //fmt格式：“YYYY-MM-DD hh:mm:ss”
        var o = {
            "M+" : this.getMonth()+1,
            "D+" : this.getDate(),
            "h+" : this.getHours(),
            "m+" : this.getMinutes(),
            "s+" : this.getSeconds()
        };
        if(/(Y+)/.test(fmt)){
            var res = String(this.getFullYear()).substr(4-RegExp.$1.length);
            fmt = fmt.replace(RegExp.$1,res);
        }
        for(var str in o){
            var reg = new RegExp('('+str+')');
            if(reg.test(fmt)){
                var res = RegExp.$1.lenth>1? ("00"+o[str]).substr(String(o[str]).length) :
o[str];
                //9===>009.substr(1)
                //0012===>0012.substr(2)
                fmt = fmt.replace(RegExp.$1,res);
            }
        }
        return fmt;
    }
}
```

```
var str = "X98Y87Z65";
// 三个数字部分加了小括号，表示子表达式
var reg = /^X(\d+)Y(\d+)Z(\d+)$/;
reg.test(str); // 此处使用exec()等其他正则表达式的匹配方法也可，下同
document.writeln(RegExp.$1); // 98
document.writeln(RegExp.$2); // 87
document.writeln(RegExp.$3); // 65
```

（三）简化DOM节点操作

```
* 面向对象的DOM操作
* show()：显示
* hide()：隐藏
* on()：事件绑定
* css()：获取/设置Css样式
* addClass()：添加类名
* removeClass()：移除类名
* attr():获取/设置html属性值
* html()：设置/获取html内容
* text()：设置/获取文本内容
```

```
class xJquery{
        // 1.this.ele获取初始化
        constructor(selector){
                // 1.1若不是传入字符串，则可能传入DOM节点（判断选择器的标签名是否存在）或类数组Array。
                // this.ele得到值，退出该函数
                if(typeof selector != 'string'){
                        if(selector.tagName){
                                this.ele = [selector];
                        }else{
                                this.ele = selector;
                        }
                return;
                }
                //1.2若传入的是选择器
        try{
            this.ele  = document.querySelectorAll(selector);//Nodelist
        }catch(error){
            var selectorName = selector.slice(1);
            if(/^#/.test(selector)){
                //变成数组为的是后面的遍历
                this.ele = [document.getElementById(selectorName)];//Array
            }else if(/^\./.test(selector)){
                try{
                    this.ele = document.getElementsByClassName(selectorName);//HTMLCollection
                }catch(err){
                    this.ele = [];
                    var res = document.getElementByTagName('*');
                    for(var i=0;i<res.length;i++){
                        var className = res[i].className.split(' ');
                        if(className.indexOf(selectorName)>=0){
                            this.ele.push(res[i]);
                        }
                    }
                }
            }else{
                //getElementsByTagName
                this.ele = document.getElementsByTagName(selector);
            }
                }
```

```
    }
    //2.显示隐藏方法
    show(){
    Array.prototype.forEach.call(this.ele,function(ele){
        ele.style.display = 'block';
    });
    // 方便链式调用
    return this;
}
hide(){
    Array.prototype.forEach.call(this.ele,function(ele){
        ele.style.display = 'none';
    });
    return this;
}
// 3.事件绑定
on(type,handler,isCapture){
    Array.prototype.forEach.call(this.ele,function(ele){
        try{
            ele.addEventListener(type,handler,isCapture)
        }catch(error){
            try{
                ele.attachEvent('on' + type,handler);
            }catch(err){
                ele['on'+type] = handler;
            }
        }
    });
    return this;
}
// 4.获取/设置Css样式
css(key,value){
    // 若没有传入value，则代表获取值，只获取第一个元素的值
    if(value === undefined){
        if(window.getComputedStyle){
            return getComputedStyle(this.ele[0])[key]
        }else if(this.ele[0].currentStyle){
            return this.ele[0].currentStyle[key]
        }else{
            return this.ele[0].style[key]
        }
    }
    // 设置，则设置所有元素的值
    Array.prototype.forEach.call(this.ele,function(ele){
        ele.style[key] = value;
    })
    return this;
}
// 5.给所有元素添加类名
addClass(name){
    Array.prototype.forEach.call(this.ele,function(ele){
        try{
            ele.classList.add(name);
```

```
            }catch(error){
                var className = ele.className.split(' ');
                if(className.indexOf(name) == -1){
                    className.push(name);
                }
                ele.className = className.join(' ');
            }
        })
    }
    // 6.移除所有元素的该类名
    removeClass(name){
        Array.prototype.forEach.call(this.ele,function(ele){
            ele.classList.remove(name)
        });
    }
    // 7.设置/获取html内容
    html(html){
        // 获取第一个元素的html内容
        if(html === undefined){
            return this.ele[0].innerHTML
        }
        // 设置所有元素的html内容
        Array.prototype.forEach.call(this.ele,function(ele){
            ele.innerHTML = html;
        });
        return this;
    }
    // 设置/获取文本内容
    text(txt){}
}
//执行$(selector)，返回实例对象
let $ = function(selector){
        return new xJquery(selector);
}
```

#

#

#

==========

#

#

#

五、第五部分：jQuery知识点

（1）补:轮播图插件

```
jQuery(function($){
        //实例对象调用源性对象里面的方法：传入对象，对象包含imgs的路径（数组存放）
        $('.box').xCarousel({
                width:320,height:320,
                type:'fade',
                imgs:['img/g1.jpg','img/g2.jpg','img/g3.jpg','img/g4.jpg','img/g5.jpg']
        });
});
```

```
//1.安全使用$,且让函数内部能使用$。函数执行完将this返回。this为（jq对象，例如该案例this为$('.box'))
(function($){
        $.fn.xCarousel = function(options){
                let defaults = {
                        width:800,
                        height:320,
                        index:0,
                        duration:1000,
                        type:'vertical',//horizontal,fade
                        imgs:[]//保存图片路径
                }
                let opt = Object.assign({},defaults,options);
                opt.len = opt.imgs.length;
                let $ul;
                let lastIndex = opt.index;
                //2.初始化方法
                let init = () => {
                        // 2.2应用插件样式
            this.addClass('xcarousel');
            this.width(opt.width);
            this.height(opt.height);
            //2.1生成元素，追加到this对象里面
            $ul = $('<ul/>');
            let $res = $.map(opt.imgs,function(item){
                let $li = $('<li/>');
                let $img = $('<img/>');
                $img.attr('src',item).appendTo($li);
                return $li;
            });
            $ul.append($res);
            $ul.appendTo(this);
            //2.4 水平滚动必须设置ul的宽度、及类名
            if(opt.type === 'horizontal'){
                $ul.addClass('horizontal');
                $ul.width(opt.width*opt.len);
            }
            //2.5 淡入淡出必须设置必须设置ul的宽高、及类名
            else if(opt.type === 'fade'){
```

```
                $ul.addClass('fade');
                $ul.css({
                    width:opt.width,
                    height:opt.height
                });
                //一开始，除了当前索引以外的其他所有li的透明度设成0
                $ul.children('li').eq(opt.index).siblings('li').css('opacity',0);
            }
            //2.3.一开始执行move，移入停止定时器，移出执行move ()
            move();
            this.on('mouseenter',()=>{
                clearInterval(this.timer);
            }).on('mouseleave',()=>{
                move();
            })
                }
                //3.move方法、show方法
                let move = ()=>{
            this.timer = setInterval(()=>{
                opt.index++;
                show();
            },opt.duration);
        };
        let show = function(){
            if(opt.index > opt.len-1){
                opt.index = 0;
            }else if(opt.index < 0){
                opt.index = opt.len-1
            }
            let obj = {};
            //当opt类型为垂直，切换top值。opt.type为水平轮播，切换left值。用对象存储，作为animate参数
            if(opt.type === 'vertical'){
                obj.top = -opt.height*opt.index;
                $ul.animate(obj)
            }else if(opt.type === 'horizontal'){
                obj.left = -opt.width*opt.index;
                $ul.animate(obj)
            }else if(opt.type === 'fade'){
                //eq()获取当前索引对应的li，透明度变成1。
                $ul.children('li').eq(opt.index).animate({opacity:1},function(){
                    lastIndex = opt.index;
                });
                //上一个索引的li，透明度变成0。执行完成后，将当前索引作为上一次索引值存储起来
                $ul.children('li').eq(lastIndex).animate({opacity:0},function(){
                    lastIndex = opt.index;
                });

            }
```

}

```
        //执行init方法
                init();
                return this;
        }
})(jQuery);
```

一、了解jQuery

（一）详细信息在jQuery的api文件

JQuery是一个兼容多浏览器的javascript类库，核心理念是write less,do more(写得更少,做得更多)。是一个快速的简洁的javascript框架，可以简化查询DOM对象、处理事件、制作动画、处理Ajax交互过程。在2006年1月由美国人John Resig在纽约的barcamp发布。

1. jQuery的构造函数和实例对象 【jQuery的对象只能用jQuery的方法】

   jQuery（“#box”）；

   得到jQuery的对象（实例）；

2. jQuery的实例属性

    1. lenght 返回jQuery对象中匹配元素的个数
    2. jQuery：当前类库版本号（一般用于判断是否是jQuery对象）
    3. 别名 $
    4. 延迟代码的执行：jQuery(document).ready(fn)
        1. 此方法传入一个匿名函数
        2. 页面DOM渲染完成时执行
        3. 简写方式： jQuery(funvtion(){});
        4. 安全使用$: jQuery(function($){});
    5. 编写jQuery的代码只需要两步
        1. 选择元素
        2. 操作元素

```
var $box = $(".box");
// $box.css("border","5px solid #58bc58");
$box.css({'border':'5px solid #58bc58',padding:'20rem'})
//获取$box下面的button
//let btns = $('button',$box);
let btns = $('button','.box');
```

二、选择器和筛选方法

1. 选择器
    1. ID选择器 $("#save");
    2. 类选择器 $(".box")

3. 标签选择器 $("div")
4. 复合选择器 $("div,span,p.myClass")
5. 属性选择器 $('[id=box]')
    1. $("li[date-inde]") 获取所有data-index属性的所有元素
    2. $('li[data-index!=10]'):data-index属性不等于10的元素,css目前未支持
    3. $('li[data-index^=10]'):data-index属性以10开头的元素
    4. $('li[data-index$=10]'):data-index属性以10结尾的元素
    5. $('li[data-index*=10]'):data-index属性包含10的元素
6. 表单选择器 $(":input")
    1. 1. :radio //匹配所有单选按钮
        2. :checkbox //匹配所有复选按钮
        3. :selected //获取已选择的option元素
        4. :checked //匹配所有被选中的元素(复选框、单选框等，select中的option)
        5. :submit //匹配所有提交按钮
        6. :reset //匹配所有重置按钮
        7. :button //匹配所有按钮
        8. :text //匹配所有的单行文本框
        9. :password //匹配所有密码框

    7.可见性

    ：hidden 匹配所有不可元素（display：none）,或者type为hidden的元素

    ：visible 匹配所有可见元素

2. 常用操作
    1. jQuery对象与原生对象的转换
        1. jQuery转原生js
            1. get(0)/[0] 获取集合中的第一个DOM节点
            2. get() 不传参数得到集合中所有的DOM节点
        2. 原生js转jQuery
            1. $(dom)
    2. 判断是否为jQuery对象

```
var box = $("#box");
if(box.jquery){

}
```

    3. 判断一个jQuery对象是否存在（是否能获取到元素）
        1. length
        2. 转成原生对象再判断

3. 筛选：利用选择器得到的结果不一定是我们想要的结果
    1. 基本筛选：

1. ： odd/:even ， :gt(n) / :lt(n) 索引支持负数
2. contains ("内容") 筛选出包含“内容” 这三个字的元素

2. 筛选的方法：
    1. first()/last() 获取集合中的第一个、最后一个元素
    2. eq(idx || -idx) 获取第idx个元素，可以是负数
    3. filter(expr| obj | ele |fn) 筛选出与指定表达式匹配的元素的集合，这个方法拥有缩小匹配的范围，用逗号分割开多个表达式
    4. map(fn) 将一组元素转换成其他数组（不论是否是元素数组）
    5. slice(start,[end]) 选取一个从start到end（不包含end）匹配的子集
    6. has(expr | ele) 保留包含特定后代的元素，去掉那些不包含指定后代的元素
    7. not(expr|ele(fn)) 删除与指定表达式匹配的元素

4. 查找 【利用当前元素去查找其他元素】
    1. 查找子元素
        1. find(expr | obj |ele) 查找后代子元素
        2. children（[expr]）:取得匹配元素的所有子元素
    2. 查找父级
        1. parent([expr]) : 获取父元素
        2. parents([expr]) 取得所有的父级元素
        3. closest([expr | obj | ele]) : 从元素本身开始，逐级向上元素匹配，并返回最先匹配的元素
        4. offsetParent（） 返回第一个有定位属性(absolute,relative,fixed)* 的父元素,如果没有定位父级，则返回html元素
    3. 查找兄弟元素
    4. next([expr]): 返回下一个同辈元素 ==> nextElementSibling
    5. prev([expr]): 获取前一个同辈元素 ==> previousElementSibling
    6. nextAll([expr]): 获取当前元素之后所有的同辈元素
    7. prevAll([expr]) 获取当前元素之前所有的同辈元素
    8. siblings([expr]) 获取当前元素的所有兄弟元素（除自身以外的所有兄弟元素 = * prevAll + nextAll）

三、jQuery动画

1. 基本动画效果
    1. 1. 显示隐藏 show() / hide()
        1. hide(duration)通过改变元素的高度、宽度、和不透明度，直到这三个属性值到0
        2. show(duration)通过改变元素的高度、宽度、和不透明度，直至内容完全可见
    2. 滑动（通过改变高度）
        1. slideDown([speed,callback]): 不断改变高度，直到样式内设定的值
        2. slideUp([speed,callback]): 不断改变高度，直到0
        3. slideToggle([speed,callback]) 当元素隐藏时调用slideDown()，当元素显示时调用slideUp()
    3. 淡入淡出
        1. fadeIn: 1)显示元素 2)不断改变透明度直到1
        2. fadeOut: 1)不断改变透明度直到0 2)隐藏元素
        3. fadeToggle([speed,callback])

4. fadeTo([[speed],opacity,[fn]]) 不断改变透明度opacity，直到设定的值，并在动画完成后可选地触发一个回调函数。

   PS： jQuery动画由三中预设速度slow，normal，fast（600,400，200）

2. 自定义动画
   1. animate（params,[speen],[fn]）
   2. :animated 获取正在执行动画的元素，一般与is()方法配合使用，用于判断元素是否处于动画状态
3. 动画队列
   1. 一个元素上的动画：
      1. 当animate中存在多个属性时，动画同时发生
      2. 当同一个元素链式调用animate时，动画是按顺序发生(队列)
   2. 不同元素上的动画：
      - 默认情况下，动画同时发生
      - 回调函数内的动画等到当前动画执行完后才接着执行
   3. stop([clearQueue],[jumpToEnd]) 不加参数：停止当前元素所有《正在运行》的动画。
      - clearQueue:值为true时，清除队列
      - jumpToEnd:值为true时，跳到当前动画的最后一帧
   4. delay(duration) 设置一个延时来推迟执行队列中之后的动画。
      - duration:延迟的时间

四、DOM节点的操作

1. 增删改
   1. 创建jQuery对象

      ```
      $('<div/>');
      $('<div>生成一个div</div>');
      ```

   2. 元素添加
      1. 内部添加（添加子元素）
         1. 1. append(content|obj|ele|fn): 在元素内部最后面追加内容（后置）
            2. prepend: 向元素内部增加内容（前置）
            3. appendTo,prependTo
      2. 外部添加（添加为兄弟元素）
         1. after: 在元素后面插入内容
         2. before: 在元素前面插入内容
         3. insertAfter,insertBefore

            备注：， appendTo, prependTo, insertBefore, insertAfter, 和 replaceAll这个几个方法成为一个破坏性操作，返回值是所有被追加的内容，

   3. 元素删除
      1. remove(); 删除元素, 虽然元素从文档中删除了，但js内部依然保留对它引用

2. empty(); 清空内容

4. 元素复制

1. clone([Event[,deepEvent]])

- Event：（true 或 false）是否复制元素的行为，默认为false
- deepEvent: （true 或 false）是否复制子元素的行为，默认为Even的值

五、盒模型属性

- offset():获取匹配元素相对于根元素的偏移量
  返回一个对象，包含当前元素的top,left值
- position():获取匹配元素相对(有定位属性)父元素的偏移量，如果没有定位父级，则相对于根元素(html)，【返回一个对象，包含当前元素的top,left值。】
- width(v) = width; //取值/赋值,当传入v时，相当于css('width',v);
- innerWidth() = width + padding; <==> clientWidth
- outerWidth() = width + padding + border; <==> offsetWidth
- outerWidth(true) = width + padding + border + margin;

六、事件

常用事件方法

- 鼠标事件
  - click([[data],fn]) //点击时触发 click = mousedown + mouseup
  - dblclick([[data],fn]) //双击事件 dblclick = 2*click
  - mousedown([[data],fn])
  - mouseup([[data],fn])
  - mousemove([[data],fn])
  - mouseout([[data],fn])
  - mouseover([[data],fn])
  - mouseenter([[data],fn]) //事件不会冒泡
  - mouseleave([[data],fn])) //事件不会冒泡
- 键盘事件
  - keydown([[data],fn]) //键盘按下时触发
  - keypress([[data],fn])//字符按键
  - keyup([[data],fn]) //键盘弹起时触发
- 表单事件
  - blur([[data],fn]) //失去焦点时触发
  - focus([[data],fn]) //获得焦点
  - change([[data],fn]) //值改变并失去焦点时触发
  - submit([[data],fn])
- 其他事件
  - resize([[data],fn]) //元素大小改变时触发

- scroll([[data],fn]) //滚动时触发

五、事件

1.jquery事件绑定与移除

- on(type,[selector],fn)
  - selector: 把本来绑定给selector的事件委托给它的父级
  - 事件命名空间, 自定义事件（对事件加以细分）
    `格式：事件类型.自定名字`
  - 一次性绑定多个事件，事件之间以空格隔开
  - 支持自定义事件的绑定
    `$(ele).on('laowang',function(){})`
    触发自定义事件：$(ele).trigger('laowang');
- off: 清除绑定事件
  - off('click');//清除当前元素的点击事件
  - off();//清除当前元素所有事件
  - off('click mouseover') 一次性清除多个事件，事件之间以空格隔开
  - off('click.output') 清除命名空间事件

2.其他事件方法

hover(enter[,leave])

- enter:鼠标移入时执行
- leave:鼠标移出时执行

  hover方法内部使用mouseenter + mouseleave来实现效果
  - trigger(type): 手动触发事件（即使事件没有发生，也能执行事件处理函数）
  - triggerHandler(type): 这个方法会触发指定的事件类型上所有绑定的处理函数。但不会执行浏览器默认行为，也不会产生事件冒泡
  - 阻止浏览器默认行为
    event.preventDefault();
  - 阻止事件传播
    event.stopPropagation();
  - 两者一起阻止：
    return false;

六、jQuery的ajax方法

- $.ajax(settings)
  - type:请求类型，默认GET
  - url:数据请求地址（API地址）
  - data:发送到服务器的数据对象，格式：{Key:value}。
  - success:请求成功时回调函数。
  - dataType:设定返回数据的格式，json, jsonp, text(默认), html, xml, script
  - async：是否为异步请求，默认true

- $.get(url,[data],[fn],[dataType]) // type:'get'
- $.post(url,[data],[fn],[dataType]) // type:'post'
- $.getJSON(url,[data],[fn]) // type:'get', dataType:'json'
- $.getScript(url,[callback]) // type:'get', dataType:'script'
- load(url,[data],[callback]) 载入远程 HTML 文件代码并插入页面中。

常用jQuery原型对象的方法

【写在jQuery原型对象中的方法，通过jQuery实例调用 】

.css(attr[,val]): 获取/改变元素style属性（内联样式）

- 取值：css(attr),css(['color','text-align']) <==> getComputedStyle[attr]
- 赋值：css(attr,val),css({attr:val});

.val(v) 获取/设置匹配表单元素的值（等同于原生js中的value属性）

取值：input.val()

赋值：input.val（v） 【v: 字符串 数组 函数，函数的内部一定要有返回值】

html() （等同于原生js中的innerHTML）

取值div.html()：取得第一个匹配元素的html内容

赋值div.html(':')：设置匹配元素的内容

text(): 取得所有匹配元素的文本内容。

addClass()/removeClass(): 添加/删除类,支持多个类同时添加或删除

- toggleClass(): 如果存在（不存在）就删除（添加）类。
- hasClass('con'): 判断当前元素是否包含con这个类，返回布尔值（不支持多个类进行判断）

eq(n) 获取第N个jquery对象（元素）,n支持负数（表示从后面查找）

index():获取当前元素在同辈元素中的索引值 【 $(this).index】;

显示、隐藏

show():显示

hide():隐藏 带参数：同时改变width，height，opacity的动画

is(expr|obj|ele|fn)

根据选择器、DOM元素或 jQuery 对象来检测匹配元素集合，其中如果有一个元素符合这个给定的选择器表达式就返回true。如果没有元素符合，或者表达式无效，都返回false。

attr(name[,val]) 设置/获取html标签属性

prop(attr[,val]) 获取/设置DOM节点属性（一般修改布尔类型属性）

获取：获取在匹配的元素集中的第一个元素的属性值。

赋值：给集合中所有元素属性赋值

```
$(':checkbox').prop('checked',function(idx,oldVal){
    return !oldVal;
}

值为函数
```

each(function(idx,ele){}) //用于遍历jquery对象

- return true;// 跳过当前循环，进入下一个循环（等效原生js中得continue）
- return false;// 退出整个each循环（等效原生js中得break）

jquery大部分方法的共性：

- 无参数时为取值，带参数时为赋值
- 取值：取得第一个匹配元素的值
- 赋值：设置所有匹配元素的值
- 隐式迭代（隐式遍历）：看不见的遍历，大部分的jquery方法都支持隐式迭代

常用jQuery静态方法

- $.each(arr|obj,callback)：通用遍历方法，用于遍历对象和数组。 【callback(idx,item)】
- $.map(arr|obj,callback)：根据现有数组生成一个新的数组，新数组的元素为callback内return的值 【callback(item,idx)】
- $.type(n)：检测参数n的数据类型
- $.makeArray(obj) //将类数组对象转换为数组。
- $.parseJSON(json) //接受一个JSON字符串，返回解析后的对象。类似原生js中的JSON.parse
- $.inArray(value,array,[fromIndex]) //确定value在数组array中的位置，从0开始计数(如果没有找到则返回 -1)，一般用于判断数组中是否包含某一字符。
- serialize()/serializeArray() : 只能在form表单中使用，并且表单元素必须有name属性

附：报错信息总结：

1.很有可能你的符号错了，中英文符号，或者多了少了括号。

```
: Invalid or unexpected token
```

2.不能读取空的内部HTML。

```
▶Uncaught TypeError: Cannot read property 'innerHTML' of null
```

这个报错是一个初学者的问题。实际上，在页面的HTML结构中，innerHTML是有实际的值并可以在console进行获取查询到。

问题：script中关于DOM部分放在和body标签前。

根据浏览器的渲染原理，HTML代码从上到下执行代码，当浏览器JS解析器解析到script并进行DOM操作，下面的DOM结构还没有进行搭建。那么就会提示不能正确读取内部的HTML信息。

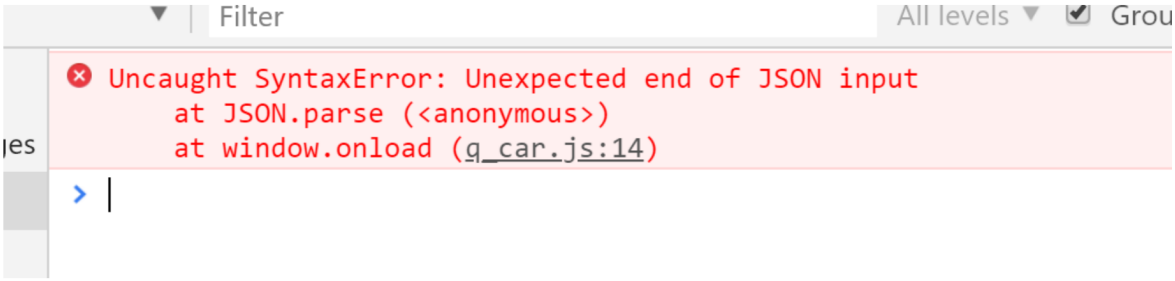
解决思路：将涉及DOM读取部分的script放在DOM结构后面。标签前面。或者在前置的script标签写明：window.onload等，确保DOM树加载完毕后再执行这段script代码。

3.JSON 能返回空字符串 “” 不能把空字符串、空数组转化为JSON

```
> JSON.parse("")
▶Uncaught SyntaxError: Unexpected end of JSON input
    at JSON.parse (<anonymous>)
    at <anonymous>:1:6
>
```

```
> JSON.parse([])
▶Uncaught SyntaxError: Unexpected end of JSON input
    at JSON.parse (<anonymous>)
    at <anonymous>:1:6
>
```

UnTySyTraceError：JSON输入的意外结束



Unexpected end of input 的英文意思是“**意外的终止输入**”

他通常表示我们浏览器在读取我们的js代码时，碰到了**不可预知的错误**，导致浏览器 无语进行下面的读取

通常造成这种错误的原因是应该**成双的符号**输入错误，比如说**“”，‘’，{}，[]。**