**JavaScript有哪几种数据类型**

简单数据类型(原始类型)：  null   undefined   number  boolean  string

复杂数据类型(引用类型)：object

**call和apply的区别**

call和apply相同点：

都是为了用一个本不属于一个对象的方法，让这个对象去执行

toString.call([],1,2,3)

toString.apply([],[1,2,3])

Object.call(this,obj1,obj2,obj3)

Object.apply(this,arguments)

 区别：

call第二个参数开始接受一个参数列表

apply第二个参数开始接受一个参数数组

**希望获取到页面中所有的checkbox怎么做**

**var** domList = **document**.getElementsByTagName(‘input’)  
**var *checkBoxList*** = [];*//返回的所有的checkbox***var *len*** = ***domList***.**length**;　　*//缓存到局部变量***while** (***len***--) {  
 **if** (***domList***[***len***].**type** == ‘checkbox’) {  
 ***checkBoxList***.push(***domList***[***len***]);  
 }  
}

**输出今天的日期，以YYYY-MM-DD的方式，比如今天是2014年9月26日，则输出2014-09-26**

<**script**>  
 **var *d*** = **new** Date();  
 *// 获取年，getFullYear()返回4位的数字 //今年：2016* **var *year*** = ***d***.getFullYear();  
 *// 获取月，月份比较特殊，0是1月，11是12月* **var *month*** = ***d***.getMonth() + 1;  
 *// 变成两位* ***month*** = ***month*** < 10 ? **'0'** + ***month*** : ***month***;  
 *// 获取日* **var *day*** = ***d***.getDate();  
 ***day*** = ***day*** < 10 ? **'0'** + ***day*** : ***day***;  
 alert(***year*** + **'-'** + ***month*** + **'-'** + ***day***);  
</**script**>

**写一个function，清除字符串前后的空格。（兼容所有浏览器）**

<**script**>  
 **if** (!String.**prototype**.trim) {  
 String.**prototype**.trim = **function**() {  
 **return this**.replace(/^\s+/, **""**).replace(/\s+$/,**""**);  
 *//\s匹配空白字符：回车、换行、制表符tab 空格* }  
 }  
 *// test the function* **var *str*** = **" \t\n test string "**.trim();  
 alert(***str*** == **"test string"**); *// alerts "true"*</**script**>

**判断一个字符串中出现次数最多的字符，统计这个次数**

<**script**>  
 **var *str*** = **'asdfssaaasasasasaa'**;  
 **var *json*** = {};  
 **for** (**var *i*** = 0; ***i*** < ***str***.**length**; ***i***++) {  
 **if**(!***json***[***str***.charAt(***i***)]){ *//str.charAt(i) =”a”* ***json***[***str***.charAt(***i***)] = 1; *//json[“a”]=1* }**else**{  
 ***json***[***str***.charAt(***i***)]++; *//让a键的值+1，也就是这个字符多出现了一次* }  
 };  
 **var *iMax*** = 0;  
 **var *iIndex*** = **''**;  
 **for**(**var *i* in *json***){  
 **if**(***json***[***i***]>***iMax***){  
 ***iMax*** = ***json***[***i***];  
 ***iIndex*** = ***i***;  
 }  
 }  
 alert(**'出现次数最多的是:'**+***iIndex***+**'出现'**+***iMax***+**'次'**);  
</**script**>

**BOM对象有哪些，列举window对象？**

 1、window对象 ，是JS的最顶层对象，其他的BOM对象都是window对象的属性；

 2、document对象，文档对象；

3、location对象，浏览器当前URL信息；

 4、navigator对象，浏览器本身信息；

5、screen对象，客户端屏幕信息；

 6、history对象，浏览器访问历史信息；

**简述readyonly与disabled的区别**

readonly只针对input(text / password)和textarea有效，

而disabled对于所有的表单元素都有效，当表单元素在使用了disabled后，当我们将表单以POST或GET的方式提交的话，这个元素的值不会被传递出去，而readonly会将该值传递出去

**请列举js数组类型中的常用方法**

|  |  |
| --- | --- |
| **方法** | **描述** |
| [concat()](http://www.w3school.com.cn/jsref/jsref_concat_array.asp) | 连接两个或更多的数组，并返回结果。 |
| [join()](http://www.w3school.com.cn/jsref/jsref_join.asp) | 把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。 |
| [pop()](http://www.w3school.com.cn/jsref/jsref_pop.asp) | 删除并返回数组的最后一个元素 |
| [push()](http://www.w3school.com.cn/jsref/jsref_push.asp) | 向数组的末尾添加一个或更多元素，并返回新的长度。 |
| [reverse()](http://www.w3school.com.cn/jsref/jsref_reverse.asp) | 颠倒数组中元素的顺序。 |
| [shift()](http://www.w3school.com.cn/jsref/jsref_shift.asp) | 删除并返回数组的第一个元素 |
| [slice()](http://www.w3school.com.cn/jsref/jsref_slice_array.asp) | 从某个已有的数组返回选定的元素 |
| [sort()](http://www.w3school.com.cn/jsref/jsref_sort.asp) | 对数组的元素进行排序 |
| [splice()](http://www.w3school.com.cn/jsref/jsref_splice.asp) | 删除元素，并向数组添加新元素。 |
| [toSource()](http://www.w3school.com.cn/jsref/jsref_tosource_array.asp) | 返回该对象的源代码。 |
| [toString()](http://www.w3school.com.cn/jsref/jsref_toString_array.asp) | 把数组转换为字符串，并返回结果。 |
| [toLocaleString()](http://www.w3school.com.cn/jsref/jsref_toLocaleString_array.asp) | 把数组转换为本地数组，并返回结果。 |
| [unshift()](http://www.w3school.com.cn/jsref/jsref_unshift.asp) | 向数组的开头添加一个或更多元素，并返回新的长度。 |
| [valueOf()](http://www.w3school.com.cn/jsref/jsref_valueof_array.asp) | 返回数组对象的原始值 |

**JavaScript中如何对一个对象进行深度clone**

<!DOCTYPE **html**>  
<**html lang="en"**>  
 <**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
  
 </**head**>  
 <**body**>  
 <**script**>  
 Object.**prototype**.clone = **function**() {  
 **var** newObj = {};  
 **for**(**var** i **in this**) {  
 **if**(**typeof**(**this**[i]) == **"object"** || **typeof**(**this**[i]) == **"function"**) {  
 newObj[i] = **this**[i].clone();  
 }  
 **else** {  
 newObj[i] = **this**[i];  
 }  
 }  
 **return** newObj;  
 };  
  
 Array.**prototype**.clone = **function**() {  
 **var** newArray = [];  
 **for**(**var** i = 0; i < **this**.**length**; i++) {  
 **if**(**typeof**(**this**[i]) == **"object"** || **typeof**(**this**[i]) == **"function"**) {  
 newArray[i] = **this**[i].clone();  
 }  
 **else** {  
 newArray[i] = **this**[i];  
 }  
 }  
 **return** newArray;  
 };  
  
 **Function**.**prototype**.clone = **function**() {  
 **var** that = **this**;  
 **var** *newFunc* = **function**() {  
 **return** that.apply(**this**, arguments);  
 };  
 **for**(**var** i **in this**) {  
 *newFunc*[i] = **this**[i];  
 }  
 **return** *newFunc*;  
 };  
  
 **function** *Owner*(name) {  
 **this**.**name** = name;  
 }  
  
 **function** *Car*(owner, brand) {  
 **this**.**owner** = owner;  
 **this**.**brand** = brand;  
 **this**.drive = **function**() {  
 alert(**this**.**owner**.name + **"开着"** + brand + **"飞奔"**);  
 };  
 }  
  
 **var *o*** = **new** *Owner*(**"我"**);  
 **var *c1*** = **new** *Car*(***o***, **"宝马"**);  
  
 ***c1***.drive();  
  
 **var *c2*** = ***c1***.clone();  
 ***c2***.**owner**.**name** = **"Wang"**;  
  
 ***c1***.drive();  
 </**script**>  
 </**body**>  
</**html**>

**如何控制alert中的换行**  
由于alert对话框里面的内容是要显示在对话框中的文本字符串,如果传入其他类型的值,会转换成字符串.使用 ‘’\n ‘’ 即可达到换行的目的：alert("Line 1\nLine 2");

**null和undefined的区别？**

null是一个表示"无"的对象，转为数值时为0；undefined是一个表示"无"的原始值，转为数值时为NaN。

当声明的变量还未被初始化时，变量的默认值为undefined。null用来表示尚未存在的对象，常用来表示函数企图返回一个不存在的对象。

undefined表示"缺少值"，就是此处应该有一个值，但是还没有定义。

**js延迟加载的方式有哪些？**

js的延迟加载有助与提高页面的加载速度，以下是延迟加载的几种方法：

* 1.使用setTimeout延迟方法的加载时间

延迟加载js代码，给网页加载留出更多时间

<**script type="text/javascript"** >  
 **function** *A*(){  
 $.post(**"/lord/login"**,{**name**:username,**pwd**:password},**function**(){  
 alert(**"Hello"**);  
 });  
 }  
 $(**function** (){  
 setTimeout(**'***A***()'**, 1000); *//延迟1秒* })  
</**script**>

2.让js最后加载

例如引入外部js脚本文件时，如果放入html的head中,则页面加载前该js脚本就会被加载入页面，而放入body中，则会按照页面从上倒下的加载顺序来运行[JavaScript](http://lib.csdn.net/base/javascript)的代码~~~ 所以我们可以把js外部引入的文件放到页面底部，来让js最后引入，从而加快页面加载速度

上述方法2也会偶尔让你收到Google页面速度测试工具的“延迟加载javascript”警告。所以这里的解决方案将是来自Google帮助页面的推荐方案。

//这些代码应被放置在</**body**>标签前(接近HTML文件底部)  
<**script type="text/javascript"**>  
 **function** *downloadJSAtOnload*() {  
 **var** element = **document**.createElement(**"script"**);  
 element.**src** = **"defer.js"**;  
 **document**.**body**.appendChild(element);  
 }  
 **if** (**window**.addEventListener)  
 **window**.addEventListener(**"load"**, *downloadJSAtOnload*, **false**);  
 **else if** (**window**.attachEvent)  
 **window**.attachEvent(**"onload"**, *downloadJSAtOnload*);  
 **else window**.onload = *downloadJSAtOnload*;  
</**script**>

**javascript里面的继承怎么实现**

1.原型链

基本思想：利用原型让一个引用类型继承另外一个引用类型的属性和方法。构造函数，原型，实例之间的关系：每个构造函数都有一个原型对象，原型对象包含一个指向构造函数的指针，而实例都包含一个指向原型对象的内部指针。

举个栗子：

<**script type="text/javascript"**>  
 **function** *SuperType*() {  
 **this**.**property** = **true**;  
 }  
 *SuperType*.**prototype**.getSuperValue = **function**() {  
 **return this**.**property**;  
 }  
 **function** *subType*() {  
 **this**.**property** = **false**;  
 }  
 *//继承了SuperType* SubType.**prototype** = **new** *SuperType*();  
 SubType.**prototype**.getSubValue = **function** (){  
 **return this**.**property**;  
 }  
 **var *instance*** = **new** SubType();  
 **console**.log(***instance***.getSuperValue());*//true*</**script**>

2.借用构造函数

基本思想：在子类型构造函数的内部调用超类构造函数，通过使用call()和apply()方法可以在新创建的对象上执行构造函数。

举个栗子：

<**script type="text/javascript"**>  
 **function** *SuperType*() {  
 **this**.**colors** = [**"red"**,**"blue"**,**"green"**];  
 }  
 **function** *SubType*() {  
 *SuperType*.call(**this**);*//继承了SuperType* }  
 **var *instance1*** = **new** *SubType*();  
 ***instance1***.**colors**.push(**"black"**);  
 **console**.log(***instance1***.**colors**);*//"red","blue","green","black"* **var *instance2*** = **new** *SubType*();  
 **console**.log(***instance2***.**colors**);*//"red","blue","green"*</**script**>

3.组合继承

基本思想：将原型链和借用构造函数的技术组合在一块，从而发挥两者之长的一种继承模式。

举个栗子：

<**script type="text/javascript"**>  
 **function** *SuperType*(name) {  
 **this**.**name** = name;  
 **this**.**colors** = [**"red"**,**"blue"**,**"green"**];  
 }  
 *SuperType*.**prototype**.sayName = **function**() {  
 **console**.log(**this**.**name**);  
 }  
 **function** *SubType*(name, age) {  
 *SuperType*.call(**this**,name);*//继承属性* **this**.**age** = age;  
 }  
 *//继承方法  
 SubType*.**prototype** = **new** *SuperType*();  
 Subtype.**prototype**.**constructor** = Subtype;  
 Subtype.**prototype**.sayAge = **function**() {  
 **console**.log(**this**.**age**);  
 }  
 **var *instance1*** = **new** *SubType*(**"EvanChen"**,18);  
 ***instance1***.**colors**.push(**"black"**);  
 consol.log(***instance1***.**colors**);*//"red","blue","green","black"* ***instance1***.sayName();*//"EvanChen"* ***instance1***.sayAge();*//18* **var *instance2*** = **new** *SubType*(**"EvanChen666"**,20);  
 **console**.log(***instance2***.**colors**);*//"red","blue","green"* ***instance2***.sayName();*//"EvanChen666"* ***instance2***.sayAge();*//20*</**script**>

4.寄生式继承

基本思想：创建一个仅用于封装继承过程的函数，该函数在内部以某种方式来增强对象，最后再像真正是它做了所有工作一样返回对象。

举个栗子:

<**script type="text/javascript"**>  
 **function** *createAnother*(original) {  
 **var** clone = object(original);  
 clone.sayHi = **function** () {  
 alert(**"hi"**);  
 };  
 **return** clone;  
 }  
 **var** person = {  
 **name**:**"EvanChen"**,  
 **friends**:[**"Shelby"**,**"Court"**,**"Van"**];  
 };  
 **var *anotherPerson*** = *createAnother*(***person***);  
 ***anotherPerson***.sayHi();*///"hi"*</**script**>

5.寄生组合式继承

基本思想：通过借用函数来继承属性，通过原型链的混成形式来继承方法

举个栗子:

<**script type="text/javascript"**>  
 **function** *SuperType*(name){  
 **this**.**name** = name;  
 **this**.**colors** = [**"red"**,**"blue"**,**"green"**];  
 }  
 *SuperType*.**prototype**.sayName = **function** (){  
 alert(**this**.**name**);  
 };  
 **function** *SubType*(name,age){  
 *SuperType*.call(**this**,name);  
 **this**.**age** = age;  
 }  
 inheritProperty(*SubType*,*SuperType*);  
 *SubType*.**prototype**.sayAge = **function**() {  
 alert(**this**.**age**);  
 }  
</**script**>

**闭包的定义及其优缺点**

闭包 是指有权访问另一个函数作用域中的变量的函数，创建闭包的最常见的方式就是在一个函数内创建另一个函数，通过另一个函数访问这个函数的局部变量.

闭包的优点:

（１）匿名自执行函数

我们知道所有的变量，如果不加上var关键字，则默认的会添加到全局对象的属性上去，这样的临时变量加入全局对象有很多坏处，  
比如：别的函数可能误用这些变量；造成全局对象过于庞大，影响访问速度(因为变量的取值是需要从原型链上遍历的)。  
除了每次使用变量都是用var关键字外，我们在实际情况下经常遇到这样一种情况，即有的函数只需要执行一次，其内部变量无需维护，  
比如UI的初始化，那么我们可以使用闭包：

<**script type="text/javascript"**>  
 **var *datamodel*** = {  
 **table** : [],  
 **tree** : {}  
 };  
  
 (**function**(dm){  
 **for**(**var** i = 0; i < dm.**table**.**rows**; i++){  
 **var** row = dm.**table**.**rows**[i];  
 **for**(**var** j = 0; j < row.**cells**; i++){  
 drawCell(i, j);  
 }  
 }  
 *//build dm.tree* })(***datamodel***);  
</**script**>

（２）缓存

再来看一个例子，设想我们有一个处理过程很耗时的函数对象，每次调用都会花费很长时间，  
那么我们就需要将计算出来的值存储起来，当调用这个函数的时候，首先在缓存中查找，如果找不到，则进行计算，然后更新缓存并返回值，如果找到了，直接返回查找到的值即可。闭包正是可以做到这一点，因为它不会释放外部的引用，从而函数内部的值可以得以保留。

<**script type="text/javascript"**>  
 **var *CachedSearchBox*** = (**function**(){  
 **var** cache = {},  
 count = [];  
 **return** {  
 attachSearchBox : **function**(dsid){  
 **if**(dsid **in** cache){*//如果结果在缓存中* **return** cache[dsid];*//直接返回缓存中的对象* }  
 **var** fsb = **new** uikit.webctrl.SearchBox(dsid);*//新建* cache[dsid] = fsb;*//更新缓存* **if**(count.**length** > 100){*//保正缓存的大小<=100* **delete** cache[count.shift()];  
 }  
 **return** fsb;  
 },  
 clearSearchBox : **function**(dsid){  
 **if**(dsid **in** cache){  
 cache[dsid].clearSelection();  
 }  
 }  
 };  
 })();  
 ***CachedSearchBox***.attachSearchBox(**"input1"**);  
</**script**>

（３） 实现封装

可以先来看一个关于封装的例子，在person之外的地方无法访问其内部的变量，而通过提供闭包的形式来访问：

<**script type="text/javascript"**>  
 **var *person*** = **function**(){  
 *//变量作用域为函数内部，外部无法访问* **var** name = **"default"**;  
  
 **return** {  
 getName : **function**(){  
 **return** name;  
 },  
 setName : **function**(newName){  
 name = newName;  
 }  
 }  
 }();  
  
 print(***person***.name);*//直接访问，结果为undefined* print(***person***.getName());*//default* ***person***.setName(**"abruzzi"**);  
 print(***person***.getName());*//abruzzi*</**script**>

闭包的缺点

１．内存消耗

  通常来说，函数的活动对象会随着执行期上下文一起销毁，但是，由于闭包引用另外一个函数的活动对象，因此这个活动对象无法被销毁，这意味着，闭包比一般的函数需要更多的内存消耗。尤其在IE浏览器中需要关注。由于IE使用非原生javascript对象实现DOM对象，因此闭包会导致内存泄露问题

２．性能问题

使用闭包时，会涉及到跨作用域访问，每次访问都会导致性能损失。因此在脚本中，最好小心使用闭包，它同时会涉及到内存和速度问题。不过我们可以通过把跨作用域变量存储在局部变量中，然后直接访问局部变量，来减轻对执行速度的影响。

**移动端click事件延迟300ms到底是怎么回事，该如何解决？**

说明：移动设备上的web网页是有300ms延迟的，往往会造成按钮点击延迟甚至是点击失效。

历史原因：2007年苹果发布首款iphone上[iOS](http://lib.csdn.net/base/ios)系统搭载的safari为了将适用于PC端上大屏幕的网页能比较好的展示在手机端上，使用了双击缩放(double tap to zoom)的方案，比如你在手机上用浏览器打开一个PC上的网页，你可能在看到页面内容虽然可以撑满整个屏幕，但是字体、图片都很小看不清，此时可以快速双击屏幕上的某一部分，你就能看清该部分放大后的内容，再次双击后能回到原始状态。 双击缩放是指用手指在屏幕上快速点击两次，iOS 自带的 Safari 浏览器会将网页缩放至原始比例。 原因就出在浏览器需要如何判断快速点击上，当用户在屏幕上单击某一个元素时候，例如跳转链接，此处浏览器会先捕获该次单击，但浏览器不能决定用户是单纯要点击链接还是要双击该部分区域进行缩放操作，所以，捕获第一次单击后，浏览器会先Hold一段时间t，如果在t时间区间里用户未进行下一次点击，则浏览器会做单击跳转链接的处理，如果t时间里用户进行了第二次单击操作，则浏览器会禁止跳转，转而进行对该部分区域页面的缩放操作。那么这个时间区间t有多少呢？在IOS safari下，大概为300毫秒。这就是延迟的由来。造成的后果用户纯粹单击页面，页面需要过一段时间才响应，给用户慢体验感觉，对于web开发者来说是，页面js捕获click事件的回调函数处理，需要300ms后才生效，也就间接导致影响其他业务逻辑的处理。

解决方案： fastclick可以解决在手机上点击事件的300ms延迟 zepto的touch模块，tap事件也是为了解决在click的延迟问题

1. tap.js解决方案

<!DOCTYPE **html**>  
<**html lang="en"**>  
 <**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
  
 </**head**>  
 <**body**>  
 <**script src="tap.js"**></**script**>  
   
 <**div id="container"**>  
 <**button id="button-1"**>Click event</**button**>  
 <**button id="button-2"**>Tap event</**button**>  
 </**div**>  
   
 <**div id="output"**></**div**>  
   
 <**script**>  
 **var *container*** = **document**.getElementById(**'container'**)  
 **var *button1*** = **document**.getElementById(**'button-1'**);  
 **var *button2*** = **document**.getElementById(**'button-2'**);  
 **var *output*** = **document**.getElementById(**'output'**);  
   
 **var *tap*** = **new** Tap(***container***);  
   
 ***button1***.addEventListener(**'click'**, *callback*, **false**);  
 ***button2***.addEventListener(**'tap'**, *callback*, **false**);  
   
 **function** *callback* (e) {  
 e.preventDefault();  
 **var** p = **document**.createElement(**'p'**);  
 p.**textContent** = **'event: '** + e.**type**;  
 ***output***.insertBefore(p, ***output***.**firstChild**);  
 }  
 </**script**>  
 </**body**>  
</**html**>

1. fastclick.js解决方法

<!DOCTYPE **html**>  
<**html**>  
 <**head**>  
 <**title**></**title**>  
 <**style**>  
 **\*** {  
 **margin**: 0;  
 }  
 .**button** {  
 **background-color**: **#3d3d3d**;  
 **border**: 0**px**;  
 **height**: 80**px**;  
 **width**: 80%;  
 **font-size**: 50**px**;  
 **margin**: 10% 0% 0% 10%;  
 **color**: **#fff**;  
 }  
 .**fu** {  
 **min-height**: 100%;  
 **min-width**: 100%;  
 **background-color**: **Black**;  
 **background**: **rgba**(0,0,0,0.4);  
 **position**: **absolute**;  
 **top**: 0;  
 **text-align**: **center**;  
 **display**: **none**;  
 }  
 .**ts** {  
 **margin**: 8% **auto**;  
 **width**: 400**px**;  
 **height**: 400**px**;  
 **top**: 59%;  
 **background-color**: **#fff**;  
 **text-align**: **center**;  
 }  
 </**style**>  
 <**script src="fastclick.js" type="text/javascript"**></**script**>  
 <**script src="jquery-1.7.2.js" type="text/javascript"**></**script**>  
 <**script type="application/javascript"**>  
 **window**.addEventListener(**'load'**, **function** () {  
 FastClick.attach(**document**.**body**);  
 }, **false**);  
 **function** *xian*() {  
  
 $(**".fu"**).show().hide(350);  
 }  
 </**script**>  
 </**head**>  
 <**body**>  
 <**div**>  
 <**div class="but"**>  
 <**input class="button" type="button" value="点击我" onclick="***xian*()**"** /></**div**>  
 <**div class="fu"** >  
 <**div class="ts"**>  
 我是浮层  
 </**div**>  
 </**div**>  
 </**div**>  
 </**body**>  
</**html**>

**说一下JSON相关的概念**

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它是基于JavaScript的一个子集。数据格式简单, 易于读写, 占用带宽小.

**JSON与XML的比较**

(1).XML的优缺点

<1>.XML的优点

A.格式统一，符合标准；  
　　B.容易与其他系统进行远程交互，数据共享比较方便。  
<2>.XML的缺点  
　　A.XML文件庞大，文件格式复杂，传输占带宽；  
　　B.服务器端和客户端都需要花费大量代码来解析XML，导致服务器端和客户端代码变得异常复杂且不易维护；  
　　C.客户端不同浏览器之间解析XML的方式不一致，需要重复编写很多代码；  
　　D.服务器端和客户端解析XML花费较多的资源和时间。

(2).JSON的优缺点

<1>.JSON的优点：

A.数据格式比较简单，易于读写，格式都是压缩的，占用带宽小；  
　　B.易于解析，客户端JavaScript可以简单的通过eval\_r()进行JSON数据的读取；  
　　C.支持多种语言，包括ActionScript, C, C#, ColdFusion, Java, JavaScript, Perl, PHP, Python, Ruby等服务器端语言，便于服务器端的解析；  
　　D.在PHP世界，已经有PHP-JSON和JSON-PHP出现了，偏于PHP序列化后的程序直接调用，PHP服务器端的对象、数组等能直接生成JSON格式，便于客户端的访问提取；  
　　E.因为JSON格式能直接为服务器端代码使用，大大简化了服务器端和客户端的代码开发量，且完成任务不变，并且易于维护。  
<2>.JSON的缺点  
　　A.没有XML格式这么推广的深入人心和喜用广泛，没有XML那么通用性；  
　　B.JSON格式目前在Web Service中推广还属于初级阶段。

**如何实现浏览器内多个标签页之间的通信？都有哪些方式？**

方法一：使用localStorage

使用localStorage.setItem(key,value);添加内容

使用storage事件监听添加、修改、删除的动作

<**script**>  
 **window**.addEventListener(**"storage"**,**function**(event){  
 $(**"#name"**).val(event.key+”=”+event.**newValue**);  
 });  
</**script**>

方法二、使用cookie+setInterval

HTML代码:

<**inputidinputid**="name"><**input type="button" id="btnOK"value="发送"**>

JS代码-页面1

<**script**>  
 $(**function**(){  
 $(**"#btnOK"**).click(**function**(){  
 **varname**=$(**"#name"**).val();  
 **document**.**cookie**=**"name="**+name;  
 });  
 });  
</**script**>

JS代码-页面2

<**script**>  
 *//获取Cookie天的内容* **function** *getKey*(key) {  
 **return JSON**.parse(**"{\""**+ **document**.**cookie**.replace(/;\s+/gim,**"\",\""**).replace(/=/gim, **"\":\""**) +**"\"}"**)[key];  
 }  
 *//每隔1秒获取Cookie的内容* setInterval(**function**(){  
 **console**.log(*getKey*(**"name"**));  
 },1000);  
</**script**>

**Html事件绑定有几种方式？**

* 直接在DOM里绑定事件：<div onclick=”test()”></div> 这种方式称为原生事件或者属性事件
* 在JS里通过onclick绑定：xxx.onclick = test
* Dom标准通过事件添加进行绑定：addEventListener("click",test, false) //第三个参数为是否支持事件捕捉
* IE事件：attachEvent("onclick",test)

**有字符串var str = 'abc345efgabcab'，请写出3条js语句分别实现如下3个功能：**

* 1）、去掉字符串中的a、b、c字符，形成结果：‘345efg’
* 2）、将字符中的数字用中括号括起来，形成结果：‘abc[3][4][5]efgabcab’
* 3）、将字符串中的每个数字乘以2，形成结果：‘abc6810efgabcab’

**如何去除一个数组里面的重复的元素**

方法一:

<**script**>  
 Array.**prototype**.method1 = **function**(){  
 **var** arr[]; *//定义一个临时数组* **for**(**var** i = 0; i < **this**.**length**; i++){ *//循环遍历当前数组   
 //判断当前数组下标为i的元素是否已经保存到临时数组   
 //如果已保存，则跳过，否则将此元素保存到临时数组中* **if**(arr1.indexOf(**this**[i]) == -1){  
 arr.push(**this**[i]);  
 }  
 }  
 **return** arr;  
 }  
</**script**>

方法二:

<**script**>  
 Array.**prototype**.method2 = **function**(){  
 **var** h{}; *//定义一个hash表* **var** arr[]; *//定义一个临时数组* **for**(**var** i = 0; i < **this**.**length**; i++){ *//循环遍历当前数组   
 //对元素进行判断，看是否已经存在表中，如果存在则跳过，否则存入临时数组* **if**(!h[**this**[i]]){  
 *//存入hash表* h[**this**[i]] = **true**;  
 *//把当前数组元素存入到临时数组中* arr.push(**this**[i]);  
 }  
 }  
 **return** arr;  
 }  
</**script**>

方法三:

<**script**>  
 Array.**prototype**.method3 = **function**(){  
 *//直接定义结果数组* **var** arr[**this**[0]];  
 **for**(**var** i = 1; i < **this**.**length**; i++){ *//从数组第二项开始循  
 //如果数组当前元素在此数组中第一次出现的位置不是i  
 //那么我们可以判断第i项元素是重复的，否则直接存入结果数组* **if**(**this**.indexOf(**this**[i]) == i){  
 arr.push(**this**[i]);  
 }  
 }  
 **return** arr;  
  
 }  
</**script**>

方法四:

<**script**>  
 Array.**prototype**.method4 = **function**(){  
 *//将数组进行排序* **this**.sort();  
 *//定义结果数组* **var** arr[**this**[0]];  
 **for**(**var** i = 1; i < **this**.**length**; i++){ *//从数组第二项开始循环遍历数组   
 //判断相邻两个元素是否相等，如果相等说明数据重复，否则将元素写入结果数组* **if**(**this**[i] !== arr[arr.**length** - 1]){  
 arr.push(**this**[i]);  
 }  
 }  
 **return** arr;  
  
 }  
</**script**>

**JavaScript中有关IE与firfox的兼容性问题你知道哪些，该怎么解决**

1. px的问题  
   obj.style.left = ”100px”;  
   obj.style.top = ”100px”;  
   千万要记得写上px，否则在Firefox上面两句无效！！
2. 1. document.form.item 问题  
   (1)现有问题：  
   现有代码中存在许多 document.formName.item(”itemName”) 这样的语句，不能在 MF 下运行  
   (2)解决方法：  
   改用 document.formName.elements[”elementName”]
3. window.event  
   (1)现有问题：  
   使用 window.event 无法在 MF 上运行  
   (2)解决方法：  
   MF 的 event 只能在事件发生的现场使用，此问题暂无法解决
4. HTML 对象的 id 作为对象名的问题  
   (1)现有问题  
   在 IE 中，HTML 对象的 ID 可以作为 document 的下属对象变量名直接使用。在 MF 中不能。  
   (2)解决方法  
   用 getElementById(”idName”) 代替 idName 作为对象变量使用。
5. 变量名与某 HTML 对象 id 相同的问题  
   (1)现有问题  
   在 MF 中，因为对象 id 不作为 HTML 对象的名称，所以可以使用与 HTML 对象 id 相同的变量名，IE 中不能。  
   (2)解决方法  
   在声明变量时，一律加上 var ，以避免歧义，这样在 IE 中亦可正常运行。  
   此外，最好不要取与 HTML 对象 id 相同的变量名，以减少错误。
6. 添加事件

IE：element.attachEvent(”onclick”, func);。   
FF：element.addEventListener(”click”, func, true)。   
通用：element.onclick=func。虽然都可以使用onclick事件，但是onclick和上面两种方法的效果是不一样 的，onclick只有执行一个过程，而attachEvent和addEventListener执行的是一个过程列表，也就是多个过程。例 如：element.attachEvent(”onclick”, func1);element.attachEvent(”onclick”, func2)这样func1和func2都会被执行。

标签的自定义属性

IE：如果给标签div1定义了一个属性value，可以div1.value和div1[”value”]取得该值。   
FF：不能用div1.value和div1[”value”]取。   
通用：div1.getAttribute(”value”)。