**前端笔记**

**JS**

**一、流程控制语句**

**程序三种基本结构--**顺序结构、选择结构、循环结构

**1、选择结构：**

**if语句**

if (判断条件) {

执行语句

}else if (判断条件) {

执行语句

}else{

执行语句

}

**switch语句**

case里必须是具体的值，但是如果表达式可以返回一个具体的值true或者false也可以是表达式，这时switch里面是true

switch(true) {

case 表达式：

执行语句

break；

case 表达式：

执行语句

break；

default：

执行语句

}

switch(表达式) {

case 取值：

执行语句

break；

case 取值：

执行语句

break；

default：

执行语句

}

**2、循环结构：**

**for语句:1初始化表达式、2控制表达式、3循环后表达式**

for(var i = 0; i < n ; i ++){

执行语句

}

**while语句**

var i= 0；

which (判断条件){

执行语句

}

**do while 语句 先执行后判断至少执行一次**

var i = 0;

do {

执行语句

}while(判断条件)

**二、数组**

1、数组的声明

①通过构造函数

var arr = new Array ();

②通过字面量

var arr = [];

**2、数组的赋值和取值**

赋值：arr [索引号] = 1；

批量赋值 for循环

取值：arr [索引号]；

**3、数组的遍历 ---- 批量取值**

for (var i = 0; i < arr.length; i ++){}

1. 防止稀疏数组 arr (arr.length) = i;

**三、函数**

1、函数的定义和调用

函数的定义

①函数的声明：function 函数名 () {

函数体（要执行的代码）

}

②函数表达式：var 函数名 = function () {

函数体

};

函数的调用：函数名 ();（要调用先引用）

2、函数的参数

有参数的函数的声明：function 函数名 (参1，参2) { //函数声明时的参数只是一个占位符，是形式参数

函数体

}

有参数的函数的调用：函数名 (参1，参2) //函数调用时传入的参数是具体的数值，是实际参数

形参和实参没有关系，只不过在调用的时候把实参的值复制了一份赋值给了形参

3、函数返回值

①return 要返回的值；（函数的返回值是什么，调用这个函数就相当于调用了什么）

②函数在执行完成return语句后便会退出函数，后面的代码不会执行

③如果只打印函数名，会通过函数名找到函数体，如果想打印的是函数的返回值，那么一定要加小括号()

4、匿名函数

匿名函数：函数定义完后就赋值给了一个变量，通过变量名来调用这个函数，所以就没有必要起名字

5、递归函数

函数自己调用自己的编程技巧

6、回调函数

被当做函数传递的函数

**四、对象**

**通过对象可以方便的对变量和函数进行管理和使用**

1、键值对：一种对应关系，通过键能够方便地找到值 key:value; 对象也是键值对

属性名：属性值、索引：元素的值

2、对象的声明

①通过构造函数

var obj = new.Object();

②通过字面量

var obj = {};

3、属性：用来描述对象的特征 一般是名词 对应变量

定义：对象.属性名 = 要赋的值；

调用：对象.属性名；

4、方法：用来描述对象的行为 一般是动词 是一种函数（属于某个对象的函数就叫方法）

定义：对象.方法名 = function(){ //函数体 }；

调用：对象.方法名；

5、访问属性的两种方式

①点语法（属性名是什么就必须写什么）：obj.属性名；

②中括号（通过字符串找属性名，字符串可以通过编程拼接）：obj["属性名"];

6、遍历的两种方式

①for循环

②for-in

for(var k in json) { 语句 }

k变量代表的是json中的各个属性（key）和 var i = 0中的i是一个意思 名字不同而已

json JavaScript Object Notation（JavaScript对象标记法）是仿照JS中对象字面量的格式去书写的一串 用来记录对象数据的**字符串**，可以用于数据传输。

7、Null

null常被用于期望一个对象，但是当前先不附加任何值的情况

undefined派生自null

undefined == null //true

undefined === null //false

**四、DOM**

1、阻止a标签的默认行为

return false;

1. 事件三要素

事件源.事件 = function（）{事件处理程序}；

3、批量操作

①批量获取元素（通过标签名） getElementsByTagName ）

②批量添加事件 for循环遍历

replace 方法

replace (要找的值，要替换的值)

function 函数名（element, 要找的值，要替换的值）{

element.className = element.className.replace(要找的值，要替换的值)

}

Document Object Model 文档对象模型，就是把HTML文档模型化，当作对象来处理

1、概念

文档(Document)：就是指HTML或者XML文件

节点(Node)：HTML文档中的所有内容都可以称之为节点，常见的节点有

元素节点 属性节点 文本节点 注释节点

元素(Element)：HTML文档中的标签可以称为元素

2、结构概念

父节点 当前节点的父级

子节点 当前节点的子级

兄弟节点 和当前节点同属一个父级的节点

3、.获取元素

getElementById

getElementsByTagName 通过标签名寻找一类元素（伪数组）

4、设置属性

元素对象.属性名 = “属性值”;

<标签 属性名=”属性值”>

5、绑定事件

事件三要素

事件源.事件 = function(){ 事件处理程序 };

对象方法中的this永远指的是该方法所属的那个对象

6、常用属性

a常用标签属性

DOM对象的属性和HTML的标签属性几乎是一致的，常用的有src、title、className、href和style

b内部文本属性

innerHTML 获取和设置标签中的内容，设置的内容会当作节点对象被解析到DOM树上

innerText 获取和设置标签中的内容，设置的内容会被当作普通文本（有兼容性问题，旧版ff不支持）

textContent获取和设置标签中的内容，设置的内容会被当作普通文本（有兼容性问题，旧版IE不支持）

c常用表单属性

常见的表单元素属性有： type、value、checked、selected、disabled

type设置input元素的类型

value设置input元素的值

checked设置input元素是否选中

selected 设置下拉列表select中的option是否被选中

disabled 设置input元素是否被禁用

获取焦点事件是onfocus 失去焦点事件是onblur

d自定义属性--必须以data-开头

getAttribute() 获取标签属性

setAttribute() 设置标签属性

removeAttribute() 移除标签属性

7、节点

childNodes //子节点

children //子元素

nextSibling //下一个兄弟节点

nextElementSibling //下一个兄弟元素 有兼容性问题

previousSibling//上一个兄弟节点

previousElementSibling //上一个兄弟元素 有兼容性问题

firstChild //第一个节点

firstElementChild //第一个子元素 有兼容性问题

lastChild //最后一个子节点

lastElementChild //最后一个子元素 有兼容性问题

parentNode //父节点 （一定是元素节点，所以无需处理）

8、样式属性

style属性是对象，style对象的属性是字符串

style只能获取和设置行内样式

JS中 - 不能作为标识符，涉及到计算的时候必须直接通过JS加样式

9、动态创建元素

①插入和移除节点

在父元素中的最后追加子元素：father.appendChild(要追加的元素);

在父元素中的某个子元素前面插入子元素：father.insertBefore(要插入的元素,插到这个元素的前面);

从父元素中移除子元素：father.removeChild(要移除的子元素);

②js动态创建结构

方式一：直接在文档中书写 document.write(“内容”)

方式二：改变元素对象内部的HTML：innerHTML=”内容”

**使用 字符串** 或者 **数组** 先拼接再用innerHTML去添加到页面

方式三：创建或克隆节点并追加，参数是一个布尔值 true表示深层复制 false是浅层复制

document.createElement()

document.cloneNode()

五、函数

1、作用域、预解析和声明提升

预解析

解析器在执行当前作用域下的代码之前有一个预解析的过程

预解析的时候 会将变量和函数的声明提升到当前作用域的顶部 然后才执行

声明提升

变量提升：解析器会将变量提升到作用域的最上面，但是变量的赋值不会提升

函数提升：解析器会率先读取函数声明，并使其在执行任何代码之前可用

另外解析器会先找var再找function，因此如果变量和函数重名后找到的函数会把先找到的变量覆盖（如果代码执行后有赋值语句，该值又会把函数覆盖）

变量作用域

最外层的是全局变量

函数内的是局部变量

特殊：函数内部不加var的是全局变量

递归概念：程序调用自身的编程技巧称为递归

回调函数：函数也是一种普通的数据类型，因此函数也可以被当作参数传递，被当作参数传递的函数叫做回调函数

1.4.2.构造函数创建对象

@this和new

利用new 关键字可以声明新的对象。new 关键字可以让构造函数中this的指向改变，并让构造函数把this返回。

@构造函数

构造函数也是函数，只不过会默认返回一个对象。

@构造函数创建对象

通过构造函数创建对象更方便（不需要创建对象并返回）。更重要的是可以通过instanceof来判断实例的类型了。

1.5.基本类型和复杂类型

1.5.1.分类

基本数据类型（值类型）：直接存储值

number、string、boolean、undefined、null（基本类型只有这五个）

复杂数据类型（引用类型）：存储引用

**三元表达式：判断条件？条件为真的结果：条件为假的结果；**

**一、批量创建对象：**

1. **工厂模式**

function Student (name, age, sex, score) {

var stu = new Object();

stu.name = name;

stu.age = age;

stu.sex = sex;

stu.score = score;

stu.sayHi = function () {

console.log("大家好我叫" + this.name + "我今年" + this.age);

};

return stu;

}

var stu = new Student("zs", 18, 1, 100);

stu.sayHi();

1. **构造函数模式**

function Student (name, age, sex, score) {

var stu = new Object(); 这回就不需要手动的创建对象了，直接使用this

this.name = name;

this.age = age;

this.sex = sex;

this.score = score;

this.sayHi = function () {

console.log("大家好我叫" + this.name + "我今年" + this.age);

};

return stu; 也不需要返回对象了,因为new 可以让this返回

}

调用：

var stu 1= new Student("zs", 18, 1, 100);

var stu 2= new Student("zs", 18, 1, 100);

stu1.sayHi();

相同的函数保存了好几份，没有必要，通过原型属性可以解决这个问题

stu2.sayHi();

console.log(stu1.sayHi === stu2.sayHi);

通过构造函数模式创建出来的对象，可以通过instanceof 来辨别具体类型

**二、原型属性：prototype**

如果希望所有的数组对象都能调用某个方法，可以在这个对象的构造函数的原型属性上绑定方法

Array.**prototype**.jump = function () {

~~console.log("去吧，皮卡丘！");~~

};

var arr = new Array();

arr.jump()

通过原型属性，把公共的方法只保存一份，可以节省资源

**原型模式：**

function Student (name, age, sex, score) {

this.name = name;

this.age = age;

this.sex = sex;

this.score = score;

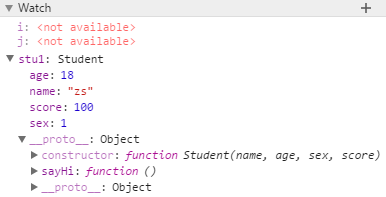
}

Student.prototype.sayHi = function () {

console.log("大家好我叫" + this.name + "我今年" + this.age);

};





调用：

var stu 1= new Student("zs", 18, 1, 100);

var stu 2= new Student("zs", 18, 1, 100);

stu1.sayHi();

stu2.sayHi();

console.log(stu1.sayHi === stu2.sayHi);

console.log(stu1);

**prototype 是构造函数的原型属性**

**\_\_proto\_\_ 是对象的原型属性,一环套一环，**

**形成了一个链条，就是原型链，通过原型链**

**可以让它现原形**

**三、深层复制数组**

function deepClone(arr) {

var newArr = [];

for (var i = 0; i < arr.length; i ++{

newArr[newArr.length] = arr[i];

}

return newArr;

[100, 2, 1]

浅层复制改变新数组会改变原数组

}

var arr1 = [3, 2, 1];

~~var arr2 = arr1;~~

[1, 2, 1]

[1, 2, 3]

深层复制改变新数组不会改变原数组

var arr2 = deepClone(arr1);

arr[0] = 100;

console.log(arr1);

console.log(deepClone(arr1).sort(function (a, b){

return a - b;

}));

**四、内置对象**

**1、Array**

var arr = [1, 2, 3];

|  |  |  |
| --- | --- | --- |
| **转换数组** | **console.log(arr.);打印输出** |  |
| console.log(arr); | [1, 2, 3] | 先以对象的形式输出，刷新后把直接量输出，其实就是调用了valueOf |
| **valueOf()** | [1, 2, 3] | 某种程度上说 它就是调用了toString |
| **toString()** | 1,2,3 | 调用了join，把每一项的值取出来用逗号拼接 |
| **join("-")** | 1-2-3 | 不传参数，默认用逗号拼接1,2,3，传入参数，会用这个参数拼接 |

|  |  |
| --- | --- |
| **检测数组** var arr = [1, 2, 3]; | **console.log打印输出的返回值** |
| **typeOf arr;** | **object** |
| **arr instanceof Array;** | **true** |
| **{} instanceof Array** | **false** |
| **Array.isArray([]);** | **true** |
| **Array.inArray({});** | **false** |
| **Array.prototype.isArray;** |  |
| **Array.isArray;** |  |
| **Ovject.prototype.toString.cell(arr);** | **[object Array]** |

|  |  |
| --- | --- |
| **增删方法** |  |
| **push(1,2,3,4);** | 从后面添加元素，可以同时加入多个 |
| console.log(arr.push(1,2,3,4)); | 返回值是新数组的长度 |
| **unshift(a);** | 从前面添加 可以添加多个 |
| console.log(arr.unshift(a)); | 返回值是新数组的长度 |
| **shift();** 无参数 | 从前面删除一个 |
| console.log(arr.shift()); | 返回删除的元素 |
| **pop()**; 无参数 | 从后面删除最后一个 |
| console.log(arr.pop()); | 返回删除的元素 |

**找到数组中第一个a的位置：arr. indexOf(a);**

**找到数组中最后一个a的位置arr. lastindexOf(a);**

找到数组中所有的a的位置

function getIndex (arr,str) {

var index = -1;

do {

index = arr.indexOf(str, index + 1);

console.log(index);

if (index ! = -1) {

console.log(index);

}

}while (index ! = -1);

}

在数组中元素出现的次数

var arr = ["c", "a", "z", "a", "x", "a"]  
 var o = {}; //属性名 元素--> 属性值 次数

for (var i = 0; i < arr.length; i ++) {

var item = arr[i];

if (o[item]) { /能进来说明有值，没有进来说明之前没有出现过

o[item] ++; //等于o[item] = o[item] + 1;

}else {

o[item] = 1; //能进来说明是undefined，也就是说之前没有出现过，这是第一次出现

}

}

操作方法：

concat();把参数拼接到当前数组

slice[begin，end);从当前数组中把要的东西复制一份，不影响原来的数组

splice(begin，删除个数，插入的新元素);删除或替换当前数组的某些元素，返回这些元素

位置方法：

indexOf(); 从前往后寻找元素在数组中的位置

lastIndexOf(); 从后往前找，没有找到返回-1

迭代方法：

every();

filter();

forEach(); 对数组遍历

for (var i = 0; i < arr.length; i ++){  
 console.log(arr[i]);}

arr.forEach(function (element, index, array) {

console.log(element);

})

map();

some();

**把原数组的值全部复制给新数组，在进行各项操作之后不会对原素组产生影响，原数组不变**

function sort(arr, fn) {

var newArr = [];

for (var i = o; i < arr.length; i ++) {

newArr[newArr.length] = arr[i];

}

for (var i = 0; i < newArr.length - 1; i ++) {

var flag = true;

for (var j = 0; j < newArr.length - 1 - i; j ++){

if (fn(newArr[j] > newArr[j + 1]) > 0){

var temp = newArr[j];

newArr[j] = newArr[j+1];

newArr[j+1] = temp;

flag = false;

}

}

if (flag) {

break;

}

}

return newArr;

}

var array = [3, 2, 1];

console.log(arr);

console.log(sort(array, function (a, b){

return a - b;

}));

console.log(arr);

**五、sort 方法（会影响原数组）**

原理：

比较的趟数

function sort(arr, fn) {

for (var i = 0; i < arr.length - 1; i ++) {

var flag = true;

两两比较的次数

for (var j = 0; j < arr.length - 1 - i; j ++){

if (fn(arr[j] > arr[j + 1]) > 0){

var temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

flag = false;

}

}

if (flag) {

break;

}

}

return arr;

}

var array = [3, 2, 1];

原数组[3,2,1]

原数组[3,2,1]

console.log(arr);

console.log(sort(array, function (a, b){

return a - b;

sort方法返回的[1,2,3]

sort方法返回的[1,2,3]

}));

console.log(arr);

sort方法执行后的原数组没有发生改变[3,2,1]

a - b从小到大排列

b - a从大到小排列

a.length - b.length 字符串从少到多排列

a.age - b.age 年龄从小到大排列

sort方法执行后的原数组发生了改变[1,2,3]

**六、形参和实参**

**调用函数的时候 是把实际参数的值 复制一份 赋值给形参 它们只是值一样 没有任何关系**

function fn(a, b) { -->形式参数

a = 100; //对形参进行改变

//console.log(a);//形参改变了

return a + b;

}

var a = 10;

var b = 20;

fn(a, b);//实际参数

//console.log(a);//实参不会改变

对形参的改变一定不会影响实参 这个说法是不严谨的

上个例子中的sort（arr,fn）

var array = [3, 2, 1];//实际参数

console.log(array);//把实际参数array复制了一份 赋值给了形式参数arr 我们是对形式参数arr进行的修改

console.log(sort(array, function (a, b) {

return a - b;

}));//这个函数内部只是对形式参数arr进行了修改

console.log(array); 实际参数也改变了

**七、基本类型与引用类型**

**基本数据类型（值类型）：直接存储值，当复制给其他参数或者做参数的时候不会改变自身的值**

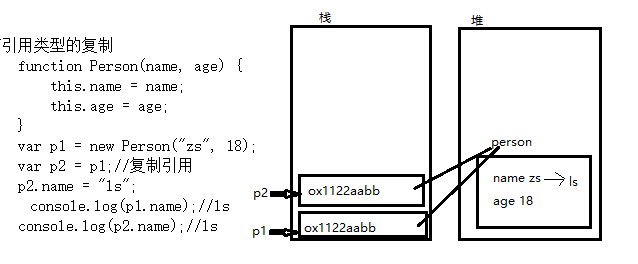
number、string、boolean、undefined、null（基本类型只有这五个）

**复杂数据类型（引用类型）：存储引用（地址），复制给其他的参数或者自己本身做参数的时候把自己储存的地址复制给了其他的参数，当其他的参数改变的时候会把地址中储存的值改变，从而改变了它自己本身**

object

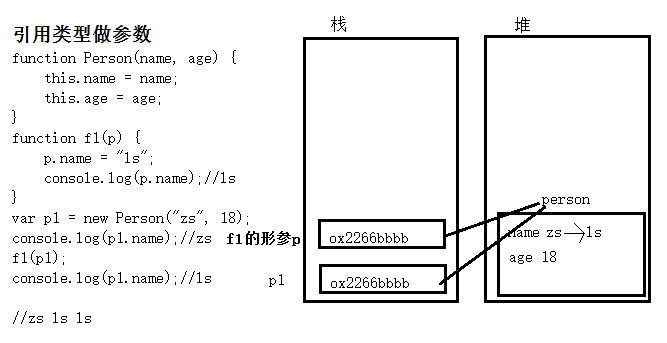
基本类型的复制

就是简单的赋值



房子

买的房子



房产证

**第三天 基础进阶**

1. **基本包装类型**

str 是基本数据类型，不是对象，怎么会有属性 str.length ???

1、创建了一个临时的包装对象 var temp = new String(str);

2、让包装对象提供我们需要的属性 console.log(temp.length);

3、使用完成后把临时对象情况 temp = null;

注意：var num = 18; //数值，基本类型

var num = Number("18"); //类型转换，基本类型

var num = new Number(18); //基本包装类型，对象

1. **构造函数**

1、Num既有数据转换的功能，又有构造函数的功能（加new）

2、new能让函数中的this返回，如果这个函数之前返回一个值类型的值，那么就不返回了

当我们new的时候，如果有return返回一个值

若返回的是值类型 new得到的就是当前构造函数本身

若返回的是引用类型，new得到的就是return的返回值

function Num(value) {

this.value = parseInt(value);

this.name = "number";

var num = parseInt(value);

return num;

}

var num1 = Num("1");

var num2 = new Num("1");

1

console.log(num1);

console.log(num2);

Num{value: 1, name: "number"}

**三、字符串的方法**

字符串的所有方法都不会修改字符串本身

**1、操作方法**

slice(参1，参2) 从start开始，截取到end位置，end取不到；参1：开始的位置 ；参2：结束的位置[start,end).遇到负数把length和负数相加

substring() 从start开始，截取到end位置，end取不到；遇到负数，把负数变为0，两个参数交换位置

substr() 从参1，start开始，截取参2，length个字符

charAt() 获取指定位置处的字符

str[0]; 和charAt 等效，IE8+支持

trim() 只能去除字符串前后的空白

split() 把字符串切割成字符数组

大小写转换：

to(Locale)UpperCase() 转换大写

to(Locale)LowerCase() 转换小写

**2、查找字符串中所有的o 的位置**

var str= "dmsosekkdsoosns";

varindex= -1；

do{

index = str.indexOf("o",index + 1);

console.log(index);

}while(index != -1);

**3、将字符串中所有的o替换成！**

do{

str = str.**replace**("o","!");

}while(str.indexOf("o")!= -1);

console.log(str);

**4、统计字符串中每个字符出现的次数**

var o = {};

for (var i = 0; i < str.length; i ++) {

var item = str.charAt(i);

if (o[item]) {

o[item] ++; //如果以前有了值，让这个值加一

}else {

o[item] = 1; //如果之前没有，现在进来就有了一次了

}

}

for (var key in o) {

console.log(key+"出现了" + o[key] + "次");

}

**5、去掉字符串中所有的空格**

var str = " aaa di djsoa jdi wd ";

str = str.trim(); //trim可以去除两边的空格

var arr = str.split(" ");//按照空格分隔

str = arr.join(" ");

**6、replaceClassName: 替换类名，封装函数**

function replaceClassName(element, oldStr, newStr) {

//把类名这个字符串按照空格分隔，把里面每一个类名做判断

var arr = element.className.split(" ");

for (var i = 0; i < arr.length; i ++) {

if (arr[i] === oldStr) {

arr[i] = newStr;

}

}

element.className = arr.join (" ");

}

**7、getElementsByClassName**

function getElementsByClassName(element,className) {

if(element.getElementsByClassName) {

return element.getElementsByClassName(className);

}else{

var filterArr = [];

var elements = element.getElementsByTagName("\*");

for (var i = 0; i < elements.length; i ++) {

var nameArr = elements[i].className.split(" ");

for (var j =0; j < nameArr.length; j ++) {

if (nameArr[j] === className) {

filterArr.push(elements[i]);

break;

}

}

}

return filterArr;}}

BOM

一、window对象

1、window.onload 加载完成事件

onload 加载完成，指的是页面所有资源都加载完成

涉及到图片内容的一定要写在onload里面

2、addLoadEvent 给同一个对象注册相同的事件会相互抵消，给window追加onload事件不会顶掉之前的函数

function addLoadEvent（fn） {

var oldOnload = window.onload;

//检查现在的window.onload是否绑定了事件

if (typeof oldOnLoad === "function") {//说明之前绑定了事件

window.onload = function () {

oldOnLoad();//之前的要执行

fn ();//传入的新的要绑定的将来也要执行

};

}else {

等同于

window.onload = function () {

fn();

}

window.onload = fn();

}

}

3、open方法

①url

②windowName窗口名

\_self 当前窗口

\_black 新窗口

③窗口的一些属性（长、宽、背景）

4、close 方法

window.close 关闭当前窗口

win.close 关闭调用它的窗口

5、setTimeout 延时定时器

setTimeout（参1，参2）参1：回调函数，要执行的代码，参2：延迟的时间,毫秒

6、clearTimeout（要清理的定时器） 清理定时器

clearTimeout（）

var timerId = null;

var btn1 = document.getElementById("btn1");

var btn2 = document.getElementById("btn2");

btn1.onclick = function () {

timerId = setTimeout(function () {

console.log("嘣")

}，3000);

};

btn2.onclick = function () {

clearTimeout(timerId);

};

7、setInterval 间歇定时器

setInterval（参1，参2）参1：回调函数，要执行的代码，参2：延迟的时间,毫秒

点击按钮，让按钮中的文字倒计时

var num = 5;

var timer = null;

btn.onclick = function () {

this.disabled = true;//1、禁用按钮

timer = setInterval(function () {//2、设置按钮的值

num--;

btn.value = num + "秒后可再次发送";//一定注意此时不能用this，定时器的回调函数是被window调用的

if (num === 0) {

clearInterval(timer);

btn.value = "点击发送验证码"；

btn.disable = false;//此时不需要再禁用了

num = 5;

}

},1000);

};

clearInterval(要清理的定时器) 清理间歇定时器

8、window的location对象

location.href = "要跳转的地址";

location.reload();让页面重新加载

location.hash; 锚点

location.host; 服务器

location.hostname; 服务器名

location.pathname

9、navigator对象

navigator.userAgent

10、history 对象

history.forward(); 前进

history.back(); 后退

11、screen 对象，屏幕对象

screen.width; 屏幕的宽度

screen.height; 屏幕的高度

screen.availWidth; 可用宽度

screen.availHeight; 可用高度

12、date日期对象

date.toString(); Sat Jul 23 2016 15:47:38 GMT+0800 （中国标准时间）

date.valueOf();这是一个13位的数字，从1970-1-1 00:00:00:000 开始到现在的毫秒值，方便运算

var date = Date.now(); 返回数值，13位的数字

var date = +new Date (); 加了+号转换成13位数字，不加就是字符串 同date.toString或console.log（date）;

可以接受三种参数：2003,10,1 日期的每一部分

"2003-10-1" 字符串的日期格式

表示日期的毫秒形式 1128094003232

var date = Date.parse("2003-10-1"); 转化成毫秒形式，格式不对返回NaN

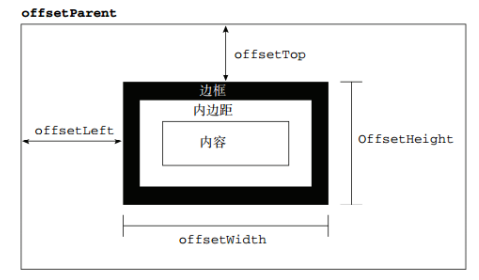
toDateString(); Sat Jul 23 2016

toTimrString(); 15:47:38 GMT+0800

toLocaleDateString(); 2016/7/23

toLocaleTimeString(); 下午3:52:11

getTine();返回毫秒数 getMilliseconds(); 当前的毫秒



getSecond(); 秒

getMinutes(); 分

getHours(); 时

getDay(); 返回周几0-6

getDate(); 返回当前月的第几天

getMonth(); 返回0-11

getFullYear(); 返回4位的年份 2016

**网页特效--01**

offset系列属性

offset -- 偏移量，相对于原来的位置移动了多少，用于获取元素真实的自身的位置和大小，是数值可以直接计算，是只读属性，无法设置

offsetWidth

offsetHeight = height + padding + border；

style 可以获取或者设置行内样式，是字符串而且有单位

offsetParent 距离自身最近的带有定位的父级，

如果父级没有定位继续往前找，直到找到body

parentNode 父节点，只和结构有关

offsetLeft 从自身的border的左侧到父级的padding左侧

一、Math对象

Math.abs();取绝对值

Math.round();就近取整 Math.round(-1.5)--> -1;

动画

原理：每次让对象向前移动一点，连起来形成动画，一秒25帧就有动画效果了

公式：对象当前位置 = 对象当前位置 + 每个时间段要移动的长度

leader = leader + step；

注意：

判断运动方向

一定要加定位

完善终点检测

终点清除定时器

手动设置对象位置到终点

调用开始先清理定时器，防止多次调用

动画函数的封装：

function animate(obj, target) {  
 clearInterval(obj.timer);

if (leader < target) {

step = step;

}else {

step = - step;

}

obj.timer = setInterval(function () {

var leader = obj.offsetLeft;

var step = 10;

step = leader < target ? step : -step;

if (Math.abs(target - leader) >= Math.abs(step)) {

leader = leader + step;

obj.style.left = leader + "px";

}else {

obj.style.left = target + "px";

clearInterval(obj.timer);

}}, 15);}

动画轮播图具体的思路：

1、要做事先找人

把所有能用到的标签全部获取到

2、动态生成结构：按钮li

2.1 根据ul 中的li 动态生成ol中的li，ul中有几个li 生成ol中几个li

2.2设置li中的文本，把li追加到ol中

2.3获取ol中的li

2.4根据第一张图，动态克隆，并追加到ul最后

3、鼠标经过按钮，移动ul到指定的位置

3.1鼠标经过按钮，按钮排他（干掉所有人，留下我自己）

3.2经过按钮，ul移动到指定位置

指定位置：-this.index \* imgWidth和当前按钮的索引以及图片的宽度有关，向左移动是负数

①设置索引②取图片的宽度③调用动画函数

4、鼠标点击箭头

4.1 显示出箭头

4.2 点击右箭头，把ul移动到指定位置

①鼠标点击事件

②ul移动到指定位置：-pic \* imgWidth 和图片索引和图片宽度有关

③判断图片移动到最后一张后（假的第一张），瞬间调到开头，然后做从第一张到第二张的动画

ul.style.left = 0; //瞬间移动到开头

pic = 0;//索引后归零，后续让ul从第一张渐渐移动到第二张

4.3点击左箭头

当图片移动到第一张后，瞬间跳到最后，然后做从最后一张到倒数第二张的动画

5、按钮也要跟着左右箭头移动

根据当前图片的索引，计算出下一个应该亮起的按钮的索引

干掉所有人、、留下相应的

6、添加自动滚动

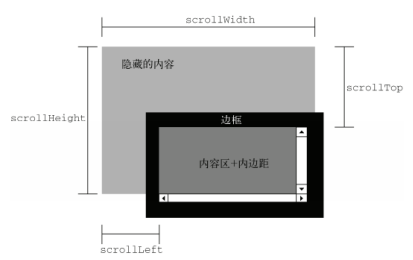
每隔一秒播放下一张（让点击右箭头的事件每隔一秒执行一次）

鼠标放在图片上自动滚动停止，离开后继续自动滚动

鼠标经过图片清除自动滚动，离开时继续自动滚动

在图片自动滚动时，经过按钮放开后，图片会接着自动滚动时的顺序继续滚动，而不是从经过按钮的图片开始滚动：把记录显示按钮的索引变为当前按钮的索引

把记录显示图片的索引变为当前图片的索引



**网页特效-2**

scroll系列属性

offset 元素自身的大小

scroll 元素内部内容的大小

scrollTop 顶部被卷去的内容的高度

scrollLeft 左侧被卷去的内容的宽度

onscroll 滚动事件

二、页面滚动坐标

var scrollTop = window.pageYoffset || document.documentElement.scrollTop || document.body.scrollTop || 0

document.body 获取的是body标签，是对象

document.head 获取的是head 标签，是对象

document.title 获取的是title 标签中的内容，是字符串

document 没有html 这个属性，html 标签是documentElement

三、封装自己的scroll

封装一个函数，调用这个函数 .top 就可以获取被卷去的头部的高度 .left 就可以获取被卷去的左侧的宽度

function scroll() {

var scrollTop = window.pageYOffset || document.documentElement.scrollTop || document.body.scrollTop || 0;

var scrollLeft= window.pageXOffset || document.documentElement.scrollLeft || document.body.scrollLeft || 0;

var o = {};

o.top = scrollTop;

o.left = scrollLeft;

return o;

}

优化方案

function scroll() {

return {

top = window.pageYOffset || document.documentElement.scrollTop || document.body.scrollTop || 0;

left= window.pageXOffset || document.documentElement.scrollLeft || document.body.scrollLeft || 0;

};

}

四、缓动动画

缓动动画公式：step = (target - leader) / 10 leader = leader + step

缓动函数的封装：让任意对象移动到指定位置

function animate(obj, target){

clearInterval(obj.timer);

obj.timer = setInterval(function(){

var leader = obj.offsetLeft;

var step = (target - leader) / 10;

step = step > 0 ? Math.ceil(step) : Math.floor(step);

leader = leader + step;

obj.style.left = leader + "px";

if (leader === target) {

clearInterval(obj.timer);

}

},15);

};

访问属性的两种方式：obj.name、obj["name"]

获取计算后样式属性：window.getComputedStyle

ie 6,7,8 :currentStyle

function getStyle (obj,attr) {

if (window.getComputedStyle) {

return window.getComputedStyle(obj,null)[attr];

} else {return obj.currentStyle[attr];}}

封装缓动框架

function animate(obj, json, fn) {

clearInterval(obj.timer);

属性名k

obj.timer = setInterval(function () {

属性值json[k]

var flag = true;

for (var k in json) { //json {attr : taget}

if (k === "opacity") { //opacity要特殊处理

//opacity没有单位 参与运算自动转换成数值 所以不用parsetInt

//取值范围 0-1 0.1 0.33 33 为了让以前的计算公式生效 要扩大100倍

var leader = getStyle(obj, k) \* 100;

var target = json[k] \* 100;

var step = (target - leader) / 10;

step = step > 0 ? Math.ceil(step) : Math.floor(step);

leader = leader + step;

obj.style[k] = leader / 100; //opacity没有单位

} else if (k === "zIndex") {

obj.style.zIndex = json[k]; //层级不需要渐变 直接设置即可

} else {

var leader = parseInt(getStyle(obj, k)) || 0;

var target = json[k];

var step = (target - leader) / 10;

step = step > 0 ? Math.ceil(step) : Math.floor(step);

leader = leader + step;

obj.style[k] = leader + "px";

}

if (leader != target) {

flag = false;

}

}

if (flag) {

clearInterval(obj.timer);

if (fn) {

fn();

}

}

}, 15);

}

//全部属性都到达目标值才能清空

function getStyle(obj, attr) {

if (window.getComputedStyle) {

return window.getComputedStyle(obj, null)[attr];

} else {

return obj.currentStyle[attr];

}

}

网页特效-3

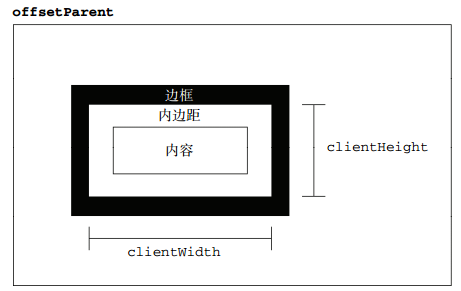
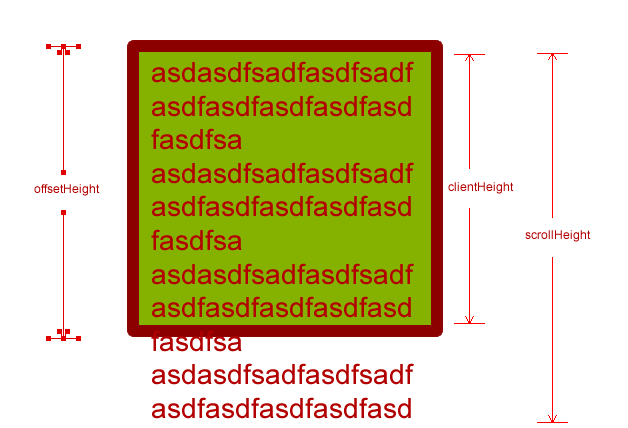
client系列

client 可视

偏移offsetWidth: width + padding + border

卷曲scrollWidth: width + padding 不包含border 内部内容的大小

可视clientWidth: width + padding 不包含border



网页可视区兼容性写法的封装（所有的浏览器）

function client() {

return {

width :window.innerWidth || document.documentElement.clientWidth || document.body.clientWidth || 0,

height :window.innerHeight || document.documentElement.clientHeight || document.body.clientHeight || 0;

}

}

window.onresize 窗体大小发生改变事件

响应式布局 a A zqQ ZQAZAzqqaaAq q Q qAA qqqqqqqqq

window.onresize = function () {

clearTimeout();

timer = setTimeout(responsive, 500)

};

respinsive();

function responsive() {

if(client()).width > 960){

document.body.style.backgroundColor = "red";

document.body.innerHTML = "computer";

} else if (client()).width > 640){

document.body.style.backgroundColor = "green";

.body.innerHTML = "tabletr";

}else {

document.body.style.backgroundColor = "yellow";

document.body.innerHTML = "mobile";

}}

函数节流

my 要使用animate 中的代码，要先加载animate

方法：windo.scrollTo(x轴坐标，y轴坐标) 窗体滚动

网页特效-4

事件对象 ：document

1、实际上，事件发生的时候，是浏览器让我们注册的这个浏览器执行的，浏览器在调用这个函数的时候会传入一个参数

2、形参和实参可以不一致，如果传入了实参没有形参接收也不会出错，只是实参没有用

二、事件对象的兼容性写法

属性

参数

document.onclick = function (event) {

局部变量

var event = event || window.event;

};

三个重要坐标

clientX clientY 在浏览器上的位置

pageX pageY 在页面上的位置 page

page 有兼容问题，client 没有，我们可以用client 来计算page = client + screen.left

var pageX = event.pageX || event.clientX + document.documentElement.scrollLeft;

var pageY = event.pageY || event.clientY + document.documentElement.scrollTop;

screenX screenY 电脑屏幕

onmousedown 鼠标按下事件

onmouseup 鼠标弹起事件

onmousemove 鼠标移动事件

event.stopPropagation();停止传播，防止冒泡

event.cancelBubble 阻止冒泡 ie678

event.target 通过事件目标可以找到事件的源头

判断是否选中文字

var txt = window.getSelection ? window.getSelection().toString() : document.selection.createRange().text;

事件委托

参1：要委托事件的父元素，参2：事件类型，参3：事件发生在哪些标签上，参4：发生事件后要执行的代码

绑定事件的两种方式：

1、普通事件

2、事件监听器

addEventListener(参1，参2，参3) 参1：事件的类型，参2：事件处理函数，参3：是否使用捕获

是否使用捕获：false 不使用捕获，是冒泡的顺序，从子级到父级

true 使用捕获，从父级到子级

解绑事件的两种方式：

直接顶掉

事件监听器的

正则表达式

声明和使用

1、通过构造函数定义

var 变量名= new RegExp(/表达式/);

2、通过直接量定义(只能使用一次)

var 变量名= /表达式/; 变量名可以是 regEx

常用方法，可用于检测传入的字符串是否符合该规则并返回布尔值  
 exp.test("要检测的字符串") true、false 可以匹配数字，字母不行，只要有想要的（数字）就是true

预定义类

. [^\n\r] 除了换行和回车之外的任意字符

\d [0-9] 数字字符digit

\D [^0-9] 非数字字符

\w [a-zA-Z0-9\_] 单词字符(所有的字母数字和\_) word

\W [^a-zA-Z0-9\_] 非单词字符

\s [\f\r\n\t\v] 不可见字符 space-->空字符串、空格、换行

\S [^\f\r\n\t\v] 可见字符

转义符

\f 表示换页 form feed

\t 表示水平制表符 table

\v 表示垂直制表符 vertical table

| 表示或， /f|boot/.test("foot");这个表达式表示的意思是 f或者 boot 只不过foot里面有了f所以会返回true

1.1.3字符类

@字符类

简单类[abc] 表示该位置可以出现的字符，/[abc]/.test("a"); 返回true，可以是a或b或c

负向类[^abc] 表示该位置不可以出现的字符

范围类[a-e] 表示该位置可以出现的字符的范围

组合类[a-xA-E0-9] 范围类的组合

@验证密码强度

边界 量词 括号

@边界

^ 会匹配行或者字符串的起始位置

^只有在[]内才表示非 在外边表示开始

$ 会匹配行或字符串的结尾位置

^$在一起 表示必须是这个（精确匹配），必须是我要的那个

@量词 自定义出现的次数放在 {} 中

"\*" 重复零次或多次 x>=0

"+" 重复一次或多次 x>=1

"?" 重复零次或一次 x=(0||1)

{n} n次 x=n

{n,} 重复n次或更多 x>=n

{n,m} 重复出现的次数比n多但比m少 n<=x<=m

@括号总结

()表示一组

[]表示一个字符的位置

{}表示次数

常见项目的匹配

常见项目的匹配网上有很多（例如搜索常用正则表达式大全），无需记忆，能看懂即可

匹配国内电话号码：@验证座机

/^0\d{2,3}-\d{7,8}$/

匹配姓名：@验证姓名 /^[\u4e00-\u9fa5]{2,}$/

匹配腾讯QQ号：

/^[1-9]\d{4,10}$/

匹配手机号：

/^(13[0-9]|14[57]|15[0-9]|18[0-9])\d{8}$/

匹配邮箱：

/^\w+([+-.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*$/

function check (inp, reg) {

inp.onblur = function () {

if (reg.test(this.value)){

this.nextSibling.innerHTML = "输入正确";

this.nextSibling.className = "";

} else {

this.nextSibling.innerHTML = "输入不正确";

}

};

}

1.1.6封装自己的trim方法

@replace() 方法

格式：

字符串对象.replace(正则式或字符串，替换的目标字符)

返回值:

替换后的新字符串

@封装自己的trim()方法

function trim(str) {

return str.replace(/^\s+/g,"").replace(/\s+$/g,"");

}

----->>

function trim(str) {

return str.replace(/^\s+|\s+$/g,"");

}

键盘事件：

onkeydown: 键盘按下，按着不放会一直执行

onkeyup: 键盘弹起

onkeypress: 键盘按下，用的是Ascll码

获取焦点：

检测按下的是哪一个键：e.keyCode

元素获取焦点：obj.focus();

**jQuery**

**jQuery第一天**

一、jQuery入口函数和js入口函数的区别

jQuery的入口函数：$(function(){}); 或者 $(document).ready(function(){}); jQuery === $;

$ 在jQuery中就是一个函数，根据传入参数类型的不同，分别进行不同的处理，来实现不同的功能

js入口函数指的是：window.onload = function() {};

jQuery对象和DOM对象的相互转换

$("#btn")本质上存储的就是DOM对象，所以可以使用[]的形式，来获取数组里面的元素

DOM对象此处指的是：使用js操作DOM返回的结果。

var btn = document.getElementById(“btnShow”); // btn就是一个DOM对象

jQkuery对象此处指的是：使用jQuery提供的操作DOM的方法返回的结果。

jQuery拿到DOM对象后又对其做了封装，让其具有了jQuery方法的jQuery对象，说白了，就是把DOM对象重新包装了一下。

DOM对象转换成jQuery对象：

var $btn1 = $(btn); // 此时就把DOM对象btn转换成了jQuery对象$btn1

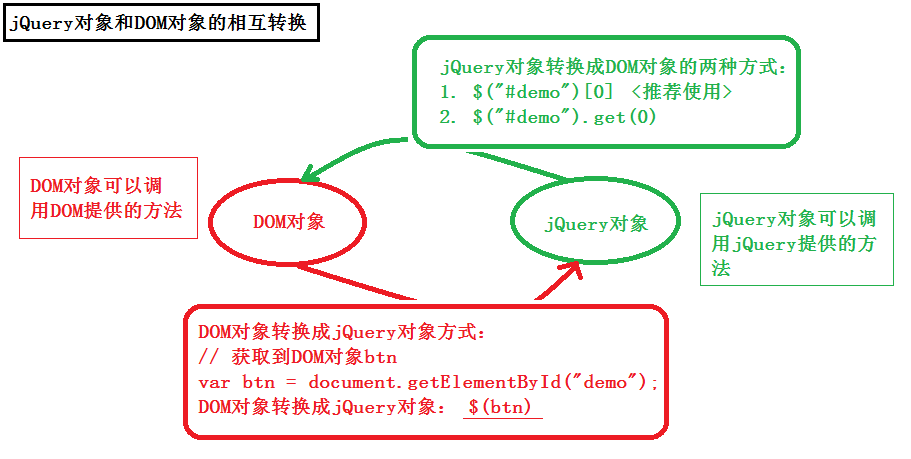
// $(document）.ready(function(){}); // 调用入口函数

// 此处是将document这个js的DOM对象，转换成了jQuery对象，然后才能调用jQuery提供的方法：ready

jQuery对象转换成DOM对象：

// 第一种方式var btn1 = $btn[0]; // 此时就把jQuery对象$btn转换成了DOM对象btn1

// 第二种方式var btn2 = $btn.get(0);// 此时就把jQuery对象$btn转换成了DOM对象btn2



jQuery选择器

1、基本选择器

|  |  |
| --- | --- |
| 符号(名称) | 说明 |
| # | Id选择器 |
| . | 类选择器 |
| element | 标签选择器 |

层级选择器

|  |  |
| --- | --- |
| 符号(名称) | 说明 |
| 空格 | 后代选择器 |
| > | 子代选择器 |
| one+two | 选取所有one元素同辈的下一个元素two（如果有几个one，就有几个two，除非紧挨着one后面没有同级元素two） |
| one~two | 选取one元素同辈的two的所有元素 |

基本过滤选择器

|  |  |
| --- | --- |
| 符号(名称) | 说明 |
| :first | 选取第一个元素 |
| :last | 选取最后一个元素 |
| :not(selector) | 选取不是这个元素的其他元素 |
| :eq(index) | 选择索引号为index的一个元素，index从0开始 |
| :odd | 选择索引号为奇数的所有元素，index从0开始 |
| :even | 选择索引号为偶数的所有元素，index从0开始 |
| :gt(index) | 选取索引大于index的元素 |
| :lt(index) | 选取索引小于index的元素 |
| :header | 选取所有的标题元素，如h1,h2,h3 |
| :animated | 选取当前正在执行动画的所有元素 |
| :focus | 选取当前获取焦点的元素 |

4、筛选选择器（方法）

|  |  |
| --- | --- |
| 符号(名称) | 说明 |
| find(selector) | 查找指定元素的所有后代元素（子子孙孙） |
| children() | 查找指定元素的直接子元素（亲儿子元素） |
| siblings() | 查找所有兄弟元素（不包括自己） |
| :parent() | 查找父元素（亲的） |
| :nth-child(index/even/odd/equation) | 选取父元素下的第index个子元素或者奇偶元素，index从1算起 |
| :first-child | 获取个父元素的第一个子元素 |
| :last-child | 选取父元素的最后一个子元素 |
| :eq(index) | 查找指定元素的第index个元素，  index是索引号，从0开始 |
| index() | 获取指定元素的索引号（从0开始的） |
| prevAll() | 获取指定元素之前的所有兄弟元素 |
| nextAll() | 获取指定元素之后的所有兄弟元素 |
| 内容过滤选择器 | |
| :contains(text) | 选取含有文本内容为“text”的元素 |
| :empty | 选取不包含子元素或者文本的空元素 |
| :has(selector) | 选取含有选择器所匹配的元素的元素 |
| :parent() | 选取含有子元素或者文本的元素 |
| 可见性过滤选择器 | |
| :hidden | 选取所有不可见的元素 |
| :visible | 选取所有可见的元素 |
| 属性过滤选择器 | |
| [ attribute ] | 选取拥有此属性的元素 |
| [ attribute = value ] | 选取属性的值为value的元素 |
| [ attribute != value ] | 选取属性的值不等于value的元素 |
| [ attribute ^= value ] | 选取以属性的值value开始的元素 |
| [ attribute $= value ] | 选取属性的值以value结束的元素 |
| [ attribute \*= value ] | 选取属性的值含有value的元素 |
| [ attribute |= value ] | 选取属性等于给定字符串或以该字符串为前缀（该字符串后跟一个连字符‘-’）的元素 |
| [ attribute ~= value ] | 选取属性的值含有value的元素 |

检查某个元素是否存在的方法：if($("#tt").length > 0) {}

**jQuery第二天**

一、样式操作

1、样式属性操作 .css()

作用：设置或获取元素的样式属性值

a设置单个样式：$(selector).css(“color”, “red”);

b设置多个样式：$(selector).css({“color”: “red”, “font-size”: “30px”})

var cssStyle = {“color”: “red”, “font-size”: “30px”};$(selector).css(cssSTyle);

c获取样式属性操作：$(selector).css(“font-size”);

2、类操作

添加类样式：

$(selector).addClass(className) 为指定元素添加类classNam

移除类样式：

$(selector).removeClass(className) 为指定元素移除类 className

$(selector).removeClass(); 不指定参数，表示移除被选中元素的所有类

判断有没有类样式：

$(selector).hasClass(calssName) 判断指定元素是否包含类 className

切换类样式：

$(selector).toggleClass(className) 为指定元素切换类，该元素有类则移除，没有指定类则添加。

二、jQuery 动画

1、隐藏显示动画

（1）show() 参数可以有两个，第一个是动画的执行时长，第二个是动画执行完成后的回调函数。

参数一可以是数值类型或者字符串类型

参数二表示：动画执行完后立即执行的回调函数

$(selector).show(2000, function() {});

hide方法作用：让匹配元素隐藏掉

2、滑入滑出动画

（1）滑入动画效果（联想：生活中的卷帘门）

作用：让元素以下拉动画效果展示出来

$(selector).slideDown(speed,callback);默认值：400毫秒（同样适用于fadeIn/slideDown/slideUp）

（2）滑出动画效果

作用：让元素以上拉动画效果隐藏起来

$(selector).slideUp(speed,callback);

（3）滑入滑出切换动画效果

$(selector).slideToggle(speed,callback);

3、淡入淡出动画

（1）淡入动画效果

作用：让元素以淡淡的进入视线的方式展示出来

$(selector).fadeIn(speed, callback);

（2） 淡出动画效果

$(selector).fadeOut(1000);作用：让元素以渐渐消失的方式隐藏起来

（3）淡入淡出切换动画效果

作用：通过改变不透明度，切换匹配元素的显示或隐藏状态

$(selector).fadeToggle('fast',function(){});

（4）改变不透明度到某个值

与淡入淡出的区别：淡入淡出只能控制元素的不透明度从 完全不透明 到完全透明；而fadeTo可以指定元素不透明度的具体值。并且时间参数是必需的！

作用：调节匹配元素的不透明度

$(selector).fadeTo(1000, .5); 参1：时长，参2：不透明值、 0全透，1全不透

有规律的体现：

jQuery提供的这几个动画效果控制的CSS属性包括：高度、宽度、不透明度。{height:400px; width:300px; opacity:.4;}这三个CSS属性的共性是：属性值只有一个，并且这个值是数值（去掉单位后）。

4、自定义动画

注意：所有能够执行动画的属性必须只有一个数字类型的值。

$(selector).animate({params},speed,callback);

作用：执行一组CSS属性的自定义动画

参1：要执行动画的CSS属性（必选）

参2：执行动画时长（可选）

参3：动画执行完后立即执行的回调函数（可选）

5、stop()停止动画

$(selector).stop(clearQueue,jumpToEnd);

参1：是否清空所有的后续动画

参2：是否立即执行完当前正在执行的动画

解释：

当调用stop()方法后，队列里面的下一个动画将会立即开始。但是，如果参数clearQueue被设置为true，那么队列面剩余的动画就被删除了，并且永远也不会执行。

如果参数jumpToEnd被设置为true，那么当前动画会停止，但是参与动画的每一个CSS属性将被立即设置为它们的目标值。比如：slideUp()方法，那么元素会立即隐藏掉。如果存在回调函数，那么回调函数也会立即执行。

注意：如果元素动画还没有执行完，此时调用sotp()方法，那么动画将会停止。并且动画没有执行完成，那么回调函数也不会被执行。

jQuery节点操作

1、操作元素节点

（1）动态创建元素

var spanNode = $(“<span>我是一个span元素</span>”);

（2）添加元素append() 参数jQuery对象

$(selector).append($node);// 在$(selector)中追加$node

作用：在被选元素的最后一个子元素（或内容）后面插入内容（页面中存在或者创建出来的元素都可以）

如果是页面中存在的元素，那么调用append()后，会把这个元素从原先的位置移除，然后再插入到新 的位置。如果是给多个目标追加一个元素，那么append()方法的内部会复制多份这个元素，然后追加到多个目标里面去。

（3）$(selector).append('<div></div>');// 在$(selector)中追加div元素，参数为htmlString

①appendTo()

$(selector).appendTo(node);把$(selector)追加到node中去

②prepend()

$(selector).prepend(node);作用：在元素的第一个子元素前面追加内容或节点

③after()

$(selector).after(node);作用：在被选元素之后，作为兄弟元素插入内容或节点

④before()

$(selector).before(node);作用：在被选元素之前，作为兄弟元素插入内容或节点

**（4）html创建元素（推荐使用，重点）**

作用：①设置内容，参数为普通字符串

②创建元素，如果传入的是符合html规则的字符串，就会创建出相应的元素

$(selector).html(‘<span>传智播客</span>’);// 动态创建元素

③获取内容$(selector).html();// 获取html内容，有什么返回什么

调用这个方法，首先会清空原来的内容，然后再设置新的内容

（5）清空元素

$(selector).empty();// 清空指定元素的所有子元素（光杆司令）// 没有参数

$(selector).html(“”);// “自杀” 把自己（包括所有内部元素）从文档中删除掉

$(selector).remove();

（6）复制元素

(selector).clone();// 复制$(selector)所匹配到的元素// 返回值为复制的新元素

总结：推荐使用html(“<span></span>”)方法来创建元素或者html(“”)清空元素

2、操作form表单

（1）属性操作

①设置属性：

$(selector).attr("title", ''美女");参1：要设置的属性名称，参2：属性值，参2不存在返回属性对应的值

$(selector).prop(); 参2不存在返回false

用法和attr 一样，设置input和button的disabled属性，以及checkbox的checked属性 、selected属性，这几种动态改变的属性必须用prop

②获取属性：

$(selector).attr(“title”);参数：要获取的属性的名称，返回要获取的属性对应的值

此时，返回指定属性的值

③移除属性：

$(selector).removeAttr(“title”); // 参数为：要移除的属性的名称

④：selected 选择selected中被选中的元素

注意：设置selected、checked、disabled要使用.prop()方法。

（2）值和内容

①val()方法：

作用：设置或返回表单元素的值，例如：input,select,textarea的值

$(selector).val();// 获取匹配元素的值，只匹配第一个元素

$(selector).val(“具体值”);// 设置所有匹配到的元素的值

另外：val()能使select、checkBox、radio相应的选项被选中 val(' 上海 ');

②text() 方法:

作用：设置或获取匹配元素的文本内容

$(selector).text();//获取操作不带参数（注意：这时候会把所有匹配到的元素内容拼接为一个字符串）

$(selector).text(“我是内容”);//参数表示要设置的文本内容，会清空原有内容

3、尺寸位置操作

（1）高度操作height() ： 作用：设置或获取匹配元素的高度值

$(selector).height(200);//带参数表示设置高度 $(selector).height();//不带参数获取高度

（2）宽度操作width() ： 作用：设置或获取匹配元素的宽度值

(selector).width(200);//带参数表示设置宽度 $(selector).width();//不带参数获取宽度

（3）坐标值操作

①offset() 作用：获取或设置元素相对于文档左上角的位置

$(selector).offset();// 无参数表示获取，返回值为：{top:num,left:num}，值是相对于document的位置

$(selector).offset({left:100, top: 150});// 设置，参数数值类型不带单位

如果没有定位，会设置position：relative，如果有非static，则不会设置，但会把原来的值给覆盖掉

注意点：设置offset后，如果元素没有定位(默认值：static)，则被修改为relative

②scrollTop() 作用：获取或者设置元素垂直方向滚动的位置

$(selector).scrollTop();// 无参数表示获取偏移

$(selector).scrollTop(100);// 有参数表示设置偏移，参数为数值类型

对scrollTop的理解：

垂直滚动条位置 是可滚动区域 在 可视区域上方的 被隐藏区域的高度。

如果滚动条在最上方没有滚动 或者 当前元素没有出现滚动条，那么这个距离为0

③scrollLeft() 作用：获取或者设置元素水平方向滚动的位置

$(selector).scrollLeft(100);

**jQuery第三天**

jQuery事件机制

jQuery的事件机制，指的是：jQuery对JavaScript操作DOM事件的封装，包括了：事件绑定、事件解绑、事件触发。

一、事件绑定

简单事件绑定 >> bind事件绑定 >> delegate事件绑定 >> on【重点】

1、简单事件绑定：VC

click() 单击事件

mouseenter() 鼠标进入事件

mouseleave() 鼠标离开事件

dbclick() 双击事件

change() 改变事件，如：文本框值改变，下拉列表值改变等

focus() 获得焦点事件

blur() 失去焦点事件

keydown() 键盘按下事件

2、on方式

语法：$(selector).on(events,[selector],[data],function());

// 第一个参数：events，绑定事件的类型

// 第二个参数：selector, 执行事件的后代元素

// 第三个参数：data，传递给处理函数的数据，事件触发的时候通过event.data来使用(用处不大)

// 第四个参数：function()，事件处理函数

绑定多个事件

$(selector).on(“click mouseenter”, function(){});

表示给$(selector)匹配的元素绑定单击和鼠标进入事件

二、事件解绑 off

off解绑on方式绑定的事件（重点）。

$(selector).off(); 解绑匹配元素的所有事件

$(selector).off(“click”); 解绑匹配元素的所有click事件,自身的和动态的。

$(selector).off(“click”,“\*\*”); 参1：要解绑的事件，参2：解绑动态创建的事件。自身绑定的事件还在

三、事件触发

事件触发的时候只会触发当前元素被执行的事件，单击按钮触发div的单击事件

1、$(selector).click(); 简单事件触发，触发 click事件

2、trigger()方法触发事件，触发浏览器的默认行为

$("button").on("click",function(){

$("div").trigger("click");

});

$(selector).trigger(“click”);参数：要触发事件的名称

3、triggerHandler触发 事件响应方法，不触发浏览器行为

$(selector).triggerHandler(“focus”);比如:文本框获得焦点是浏览器的默认行为

四、事件对象

$(function(event){}): event 事件对象

event.data 传递给事件处理程序的额外数据

event.delegateTarget 代理绑定事件的元素

event.currentTarget 绑定事件的元素，等同于this，this：哪个元素触发的事件，this就指向这个元素

event.target  触发事件的元素，不一定===this

event.pageX 鼠标相对于页面的位置

event.clientX  距离浏览器的位置

event.screenX 相对于电脑屏幕

event.offsetX 相对于定位的父元素，或者body

event.stopPropagation()； 阻止事件冒泡

event.preventDefault();  阻止浏览器默认行为

event.type 事件类型：click，dbclick…

event.which 鼠标的按键类型：左1 中2 右3

event.keyCode 键盘按键代码

事件执行的顺序：

如果是单击的按钮，首先会触发按钮的单击事件，然后再触发div的单击事件

阻止事件冒泡：

event.stopPropagation()； 阻止事件冒泡

return false; 同时具有阻止冒泡和阻止默认行为的功能

五、链式编程

链式编程代码示例：

$(“li”).parent(“ul”).parent(“div”).siblings(“div”).children(“div”).html(“内容”);

链式编程原理：return this;

通常情况下，只有设置操作才能把链式编程延续下去。因为获取操作的时候，会返回获取到的相应的值，无法返回 this。

end(); // 结束当前链最近的一次过滤操作，并且返回匹配元素之前的一次状态

六、隐式迭代

// 设置操作$(“div”).css(“color”, “red”);

// 获取操作$(“div”).css(“color”); // 返回第一个元素的值

1、隐式迭代的意思是：在方法的内部会为匹配到的所有元素进行循环遍历，不用我们再进行循环

如果获取的是多元素的值，大部分情况下返回的是第一个元素的值。

2、each方法

大部分情况下是不需要使用each方法的，因为jQuery的隐式迭代特性。

如果要对每个元素做不同的处理，这时候就用到了each方法

作用：遍历jQuery对象集合，为每个匹配的元素执行一个函数

$(selector).each(function(index,element){});参1:当前元素在所有匹配元素中的索引号,参2：当前元素（DOM对象）

七、多库共存（了解）

此处多库共存指的是：jQuery占用了$ 和jQuery这两个变量。当在同一个页面中引用了jQuery这个js库，并且引用的其他库（或者其他版本的jQuery库）中也用到了$或者jQuery这两个变量，那么，要保证每个库都能正常使用，这时候就有了多库共存的问题。

// 模拟另外的库使用了 $ 这个变量

// 此时，就与jQuery库产生了冲突

var $ = { name : “itecast” };

解决方式：$.noConflict();

// 作用：让jQuery释放对$的控制权，让其他库能够使用$符号，达到多库共存的目的。此后，只能使用jQuery来调用jQuery提供的方法

八、jQuery插件机制

第三方插件

jQuery.color.js

animate()自定义动画：不支持背景的动画效果

animate动画支持的属性列表

使用步骤：

1.引入jQuery文件

2.引入插件

3.使用插件

全局jQuery函数扩展方法

$.pluginName = function() {};

jQuery对象扩展方法

$.fn. pluginName = function() {};

**HTML5**

**一、HTML5新增标签和重新定义的标签**

1、结构标签

块级元素，有意义的div

<article> 定义一篇文章，强调独立性

<header> 定义一个页面或一个区域的头部  
 <nav> 定义导航链接

<section> 定义一个区域，例如将一块内容分成几段

<aside> 定义页面内容部分的侧边栏  
 <hgroup> 定义文件中一个区块的相关信息，里面放h系列的标签，最好h3

<figure> 定义一组媒体内容以及它们的标题

<figcaption> 定义figure元素的标题

<footer> 定义一个页面或一个区域的底部

<dialog> 定义一个对话框类似微信

2、多媒体标签

<video> 定义一个视频

<audio> 定义音频内容

<source> 定义媒体资源

<canvas> 定义图片

<embed> 定义外部的可交互的内容或插件 比如 flash

3、Web 应用标签

<menu> 命令列表

火狐浏览器支持

<menuitem> menu命令列表标签（嵌入系统）

<command> menu标记定义一个命令按钮

<meter> 实时状态显示：气压、气温（ 状态标签）

属性：min、max、value、low、high value在low和high中间正常，之外会变色警告

<progrss> 任务过程：安装、加载（状态标签）进度条

<datalist> 为input标记定义一个下拉列表，配合option

<detalis> 标记定义一个元素的详细内容，配合dt、dd，有下拉框

4、注释标签

<ruby> 定义注释或音标

<rp> 告诉那些不支持Ruby元素的浏览器如何去显示，注释括号

<rt> 定义对ruby的注释内容文本，注释在ruby上面就像汉语汉字上的拼音

5、其他标签

<keygen> 定义表单里一个生成的键值（加密信息传送）

<mark> 定义有标记的文本（黄色选中状态）

<output> 定义一些输出类型，计算表单结果配合oninput事件

6、重新定义的标签

<dd> 问题

<dt> 描述

<hr> 表示主题结束，是一条水平线

<small> 表示小字体，例如注释或者法律条款

<menu> 定义用户界面的菜单，配合commond或者menuitem使用

二、HTML5视频、音频播放事件属性与API控件

1、video、audio标签

<video src = "movie.mp4" controls = "controls"><video>

或者<video controls = "controls">

<source src = "movie.mp4">

<video>

|  |  |  |
| --- | --- | --- |
| 属性 | 值 | 描述 |
| autoplay | autoplay（可省略） | 视频自动播放 |
| controls | controls（可省略） | 向用户显示播放控件 |
| width | px | 播放器宽度 |
| height | px | 播放器高度 |
| loop | loop、数字 | 播放完是否继续播放、播放次数 |
| preload | proload | 是否等待加载完再播放 |
| src | url | 视频url地址 |
| poster | imgurl | 加载等待的画面图片 |
| autobuffer | autobuffer | 设置为浏览器缓冲方式，不设置autoplay时有效 |

2、HTML5视频API 控件

（1）获得video标签

①通过DOM对象 var video = document.getElementById("videoID");

②通过jQuery 的方法 var video = $("#videoID")[0];

video标签的属性

载入视频：load()

播放视频：play()

暂停:pause()

快进10秒：currentTime +=10

播放速度增加：playbackRate ++

播放速度增加0.1：playbackRate += 0.1

音量增加：volume += 0.1

静音：muted = true

（3）事件

canplay

duration 媒体长度

timeupdate 媒体当前位置

三、表单

1、表单输入类型

|  |  |  |
| --- | --- | --- |
| 类型 | 使用示例 | 含义 |
| email | <input type="email"> | 输入邮箱格式 |
| tel | <input type="tel"> | 输入手机号码格式 |
| url | <input type="url"> | 输入url格式 |
| number | <input type="number"> | 输入数字格式 |
| search | <input type="search"> | 搜索框（体现语义化） |
| range | <input type="range"> | 自由拖动滑块 |
| color | <input type="color"> | 拾色器 |
| time | <input type="time"> | 小时 |
| date | <input type="date"> | 年月日 |
| datetime | <input type="datetime"> | 时间输入框 |
| month | <input type="month"> | 年月 |
| week | <input type="week"> | 年周 |

2、表单元素

|  |  |
| --- | --- |
| 元素 | 含义 |
| <datalist> | 数据列表 |
| <keygen> | 生成加密字符串 |
| <output> | 输出结果 |
| <meter> | 度量器 |

3、表单属性

|  |  |  |
| --- | --- | --- |
| 属性 | 用法 | 含义 |
| placeholder | <input type="text" placeholder="请输入用户名"> | 占位符 |
| autofocus | <input type="text" autofocus> | 自动获得焦点 |
| multiple | <input type="file" multiple> | 多文件上传 |
| autocomplete | <input type="text" autocomplete="off"> 值：off、on | 自动完成 |
| form | <input type="text" form="某表单ID"> |  |
| novalidate | <form novalidate></form> | 关闭验证 |
| required | <input type="text" required> | 必填项 |
| pattern | <input type="text" pattern="\d"> | 自定义验证 |

四、DOM扩展

1、获取元素

①document.getElementsByClassName ('class') 通过类名获取元素，以伪数组形式存在。

②document.querySelector('selector') 通过CSS选择器获取元素，符合匹配条件的第1个元素。

③document.querySelectorAll('selector') 通过CSS选择器获取元素，以伪数组形式存在。

2、类名操作

①Node.classList.add('class') 添加class

②Node.classList.remove('class') 移除class

③Node.classList.toggle('class') 切换class，有则移除，无则添加

④Node.classList.contains('class') 检测是否存在class

Node指一个有效的DOM节点，是一个通称。

3、自定义属性

在HTML5中我们可以自定义属性，其格式如下data-\*=""，例如

data-info="我是自定义属性"，通过Node.dataset['info'] 我们便可以获取到自定义的属性值。

Node.dataset是以对象形式存在的，当我们为同一个DOM节点指定了多个自定义属性时，Node.dataset则存储了所有的自定义属性的值。

假设某元素 <div id="demo" data-name="itcast" data-age="10">

var demo = document.querySelector('#demo');

1、读取自定义属性 demo.dataset（所有的属性及属性值）或者 demo.dataset['age']（age属性的值）

2、设置demo.dataset['name'] = 'web developer'

五、新增API

1、网络状态

window.online 用户网络连接时被调用

window.offline 用户网络断开时被调用

2、全屏  
 Node.requestFullScreen() 开启全屏显示

document.cancelFullScreen() 关闭全屏显示，只能通过document才能关闭  
 document.fullScreen检测当前是否处于全屏

全屏伪类选择器

:full-screen .box {}、:-webkit-full-screen {}、:moz-full-screen {}

3、文件读取

通过FileReader对象我们可以读取本地存储的文件，使用 [File](https://developer.mozilla.org/zh-CN/docs/DOM/File" \t "_blank" \o "此页面仍未被本地化, 期待您的翻译!)对象来指定所要读取的文件或数据。其中File对象可以是来自用户在一个 [<input>](https://developer.mozilla.org/zh-CN/docs/HTML/Element/input" \t "_blank" \o "This article hasn't been written yet. Please consider contributing!)元素上选择文件后返回的[FileList](https://developer.mozilla.org/zh-CN/docs/DOM/FileList" \t "_blank" \o "一个FileList对象通常来自于一个HTML input元素的files属性,你可以通过这个对象访问到用户所选择的文件.该类型的对象还有可能来自用户的拖放操作,查看DataTransfer对象了解详情.)对象，也可以来自由拖放操作生成的  [DataTransfer](https://developer.mozilla.org/zh-CN/DragDrop/DataTransfer" \t "_blank" \o "En/DragDrop/DataTransfer)。

实例化一个读取器：var reader = new FileReader();

读取文件reader.readAsDateURL();

reader.readAsBinaryString();

reader.readAsTEXT();  
 事件监听onload 当文读取完成时调用

属性result 文件读取结果

4、拖拽

①拖拽元素：页面中设置了draggable="true"属性的元素，其中<img>、<a>标签默认是可以被拖拽的

②目标元素：页面中任意的元素

③事件监听：根据元素类型不同，需要设置不同的事件监听

a拖拽元素

ondrag 应用于拖拽元素，整个拖拽过程都会调用

ondragstart 应用于拖拽元素，当拖拽开始时调用，只触发一次

ondragleave 应用于拖拽元素，当鼠标离开拖拽元素时调用

ondragend 应用于拖拽元素，当拖拽结束时调用

b目标元素

ondragenter 应用于目标元素，当拖拽元素进入时调用

ondragover 应用于目标元素，当停留在目标元素上时调用，

ondrop 应用于目标元素，当在目标元素上松开鼠标时调用

ondragleave 应用于目标元素，当鼠标离开目标元素时调用

5、地理定位

①获取当前地理信息（只获取一次）

navigator. geolocation.getCurrentPosition(successCallback, errorCallback, options)

②重复获取当前地理信息（多次）

navigator. geolocation.watchPosition(successCallback, errorCallback, options)

当成功获取地理信息后，会调用succssCallback，并返回一个包含位置信息的对象position。

position.coords.latitude纬度

position.coords.longitude经度

position.coords.accuracy精度

position.coords.altitude海拔高度

当获取地理信息失败后，会调用errorCallback，并返回错误信息error

可选参数 options 对象可以调整位置信息数据收集方式

a) enableHighAccuracy 高精度模式 true、false

b) timeout 超时设置，单位为ms

c) maximumAge表示浏览器重新获取位置信息的时间间隔，单位为ms

6、历史管理

提供window.history，对象我们可以管理历史记录，可用于单页面应用，Single Page Application，可以无刷新改变网页内容。

①pushState(data, title, url) 追加一条历史记录

data对象，用于存储自定义数据，通常设为null

title网页标题，基本上没有被支持，一般设为空

url 以当前域为基础增加一条历史记录，不可跨域设置

②replaceState(data, title, url) 与pushState()基本相同，不同之处在于replaceState()，只是替换当前url，不会增加/减少历史记录。

③事件监听

onpopstate事件，当前进或后退时则触发，通过事件对象ev.state可以读取到存储的数据。

7、Web存储（本地存储）

① a、设置、读取方便

b、容量较大，sessionStorage约5M、localStorage约20M

c、只能存储字符串，可以将对象JSON.stringify() 编码后存储

②window.sessionStorage

a、生命周期为关闭浏览器窗口

b、在同一个窗口下数据可以共享

③window.localStorage

a、永久生效，除非手动删除

b、可以多窗口共享

④方法详解

setItem(key, value) 设置存储内容

getItem(key) 读取存储内容

removeItem(key) 删除键值为key的存储内容

clear() 清空所有存储内容

key(n) 以索引值来获取存储内容

⑤其它

WebSQL、IndexDB

8、应用缓存

HTML5中我们可以轻松的构建一个离线（无网络状态）应用，只需要创建一个cache manifest文件。

①优势

a、可配置需要缓存的资源

b、网络无连接应用仍可用

c、本地读取缓存资源，提升访问速度，增强用户体验

d、减少请求，缓解服务器负担

②缓存清单

一个普通文本文件，其中列出了浏览器应缓存以供离线访问的资源，推荐使用.appcache为后缀名，添加MIME类型

AddType text/cache-manifest .appcache

例如我们创建了一个名为demo.appcache的文件，然后在需要应用缓存在页面的根元素(html)添加属性manifest="demo.appcache"，路径要保证正确。

③manifest文件格式

a、顶行写CACHE MANIFEST

b、CACHE: 换行 指定我们需要缓存的静态资源，如.css、image、js等

c、NETWORK: 换行 指定需要在线访问的资源，可使用通配符

d、FALLBACK: 换行 当被缓存的文件找不到时的备用资源 ./online.html ./offline.html

当online没有用offline替换

④其他

b、可以指定多个CACHE: NETWORK: FALLBACK:，无顺序限制

c、#表示注释，只有当demo.appcache文件内容发生改变时或者手动清除缓存后，才会重新缓存。

d、chrome 可以通过chrome://appcache-internals/工具和离线（offline）模式来调试管理应用缓存

9、多媒体

CSS3

一、选择器

1、属性选择器

①E[attr] 表示存在attr属性即可；

②E[attr=val] 表示属性值完全等于val；

③E[attr\*=val] 表示在属性值的“任意”位置包含val字符；

④E[attr^=val] 表示在属性值的“开始”位置包含val字符；

⑤E[attr$=val] 表示在属性值的“结束”位置包含val字符；

2、伪类选择器

①以某元素相对于其父元素或兄弟元素的位置。

E:first-child第一个子元素

E:last-child最后一个子元素

E:nth-child(n) 第n个子元素

E:nth-last-child(n) 同E:nth-child(n) 相似，只是倒着计算

②目标伪类选择器

E:empty 选中没有任何子节点的E元素；（使用不是非常广泛）

E:target 结合锚点进行使用，处于当前锚点的元素会被选中；

E:not () 选中除了当前锚点的其他元素

3、伪元素选择器

E::first-letter文本的第一个单词或字（如中文、日文、韩文等）；

E::first-line 文本第一行；

E::selection 可改变选中文本的样式；

是一个行内元素，需要转换成块元素

":" 与 "::" 区别在于区分伪类和伪元素,在开发中使用":";

二、颜色

RGBA、HSLA，可以设置颜色值的透明度，相较opacity，它们不具有继承性，即不会影响子元素的透明度。

H 色调 取值范围0~360，0/360表示红色、120表示绿色、240表示蓝色

S 饱和度 取值范围0%~100%

L 亮度 取值范围0%~100%

A 透明度 取值范围0~1

关于透明度：

1、opacity只能针对整个盒子设置透明度，子盒子及内容会继承父盒子的透明度；

2、transparent 不可调节透明度，始终完全透明background-color：transparent

三、文本

text-shadow，可分别设置偏移量、模糊度、颜色（可设透明度）。

参1：水平偏移量 正值向右 负值向左；

参2：垂直偏移量 正值向下 负值向上；

参3：模糊度（模糊半径）是不能为负值；

参4：颜色

四、边框

边框圆角：border-radius：0 0 0 0 / 0 0 0 0; 前四个是x轴圆角半径，后面四个是Y轴圆角半径

边框阴影：border-shadow

参1、水平偏移量 右正左负；

参2、垂直偏移量 下正上负；

参3、模糊半径是不能为负值；

参4、颜色

参5、inset可以设置内阴影；

边框图片：border-image

参1：source：图片路径

参2：slice：切割，包含4参数（a,b,c,d）按照上右下左顺序

参3：repeat是否平铺 包含三个值：stretch拉伸（默认值）、repeat重复、round平铺

参4：width宽度

参5：偏移位置

五、盒模型

CSS3中可以通过box-sizing 来指定盒模型，即可指定为content-box、border-box，这样我们计算盒子大小的方式就发生了改变。

可以分成两种情况：

1、box-sizing: border-box 计算方式为content = width – border - padding

2、box-sizing: content-box 计算方式为content = width

box-shadow属性

取值：水平偏移量、垂直偏移量、模糊度、外延值、color、inset（内阴影）

box-reflect盒子倒影属性:可以对盒子模型进行倒影设置。

值： direction（above、below、left、right）、offset（图片与倒影间隔）、mask-box-image(url)

六、背景

背景在CSS3中也得到很大程度的增强，比如背景图片尺寸、背景裁切区域、背景定位参照点、多重背景等。

1、background-size设置背景图片的尺寸

cover会自动调整缩放比例，保证图片始终填充满背景区域，如有溢出部分则会被隐藏。

contain会自动调整缩放比例，保证图片始终完整显示在背景区域。

也可以使用长度单位或百分比

2、background-origin设置背景定位的原点

border-box 以边框做为原点；

padding-box 以内边距做为原点；

content-box 以内容区做为原点；

3、background-clip设置背景图片的裁切区域

border-box 裁切边框以内为背景区域；

padding-box裁切内边距以内为背景区域；

content-box 裁切内容区做为背景区域；

background-break 设置背景图片进行平铺时的循环方式

bounding-box 在整个元素中进行平铺

each-box 在每一行中进行平铺

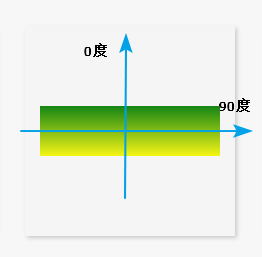
continuous 下一行中的图像紧接着上一行中的图像继续平铺

七、渐变

线性渐变

linear-gradient线性渐变指沿着某条直线朝一个方向产生渐变效果。

取值：方向、起止色、渐变范围



2、径向渐变

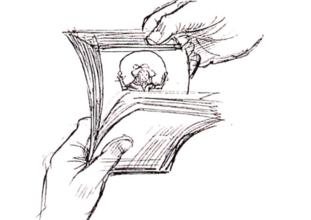
radial-gradient径向渐变指从一个中心点开始沿着四周产生渐变效果

取值：圆半径、圆中心、起止色、渐变范围

八、过渡

过渡可以实现元素不同状态间的平滑过渡（补间动画），经常用来制作动画效果。

帧动画：通过一帧一帧的画面按照固定顺序和速度播放。如电影胶片



补间动画：自动完成从起始状态到终止状态的的过渡。

特点：当前元素只要有“属性”发生变化时，可以平滑的进行过渡。e

过渡属性在A、B状态都可以，如果都要慢慢变化就设在A上,hover或者JS添加class

简写：第一个时间是执行时间，第二个是延迟时间，其他的没有顺序要求，但是最好按照下面的顺序书写

transition-property设置过渡属性

取值：none 没有元素

all 所有元素

index 指定的css属性

transition-duration设置过渡时间，从旧属性转换到新属性花费的事件

transition-delay设置过渡动画延迟时间

transition-timing-function设置过渡速度（几个取值肉眼很难看出来区别）

取值：ease 缓解效果，等同于cubic-bezier(0.25, 0.1, 0.25, 1.0)函数，即立方贝塞尔

linear 线性效果，等同于cubic-bezier(0.0, 0.0, 0.1, 1.0)函数

ease-in 渐显效果，等同于cubic-bezier(0.42, 0, 1.0, 1.0)函数

ease-out 渐隐效果，等同于cubic-bezier(0, 0, 0.58, 1.0)函数

ease-in-out 渐显渐隐效果，等同于cubic-bezier(0.42, 0, 0.58, 1.0)函数

cubic-bezier 特殊的立方贝塞尔曲线

九、转换

1、2D转换

transform：用来设置2D或3D转换

转换可以实现元素的位移、旋转、变形、缩放，甚至支持矩阵方式

①移动 translate(x, y) 改变元素的位置，相对原来的位置的变化，如果为百分比则是相对于盒子的大小；

translate3d(x,y,z) 3d空间的位置移动

②缩放 scale(x, y) 水平和垂直方向的缩放，x、y的取值可为小数，如果只有一个参数即缩放的倍数；

③旋转 rotate(deg) 可以对元素进行旋转，正值为顺时针，负值为逆时针；旋转时元素的坐标也会跟着改变

④倾斜 skew(Ydeg, Xdeg) 可以使元素按一定的角度进行倾斜，会改变元素的形状

⑤矩阵变换matrix(x, y) 改变元素的位置

2、3D转换

（1）3D坐标轴

用X、Y、Z分别表示空间的3个维度，三条轴互相垂直。如下图



（2）3D呈现

transform-style:preserve-3D; 内嵌元素，并且必须经过Z变形（transform）

透视

perspective :辅助性帮我们去理解元素在转换过程中的视觉效果

(0-1000)设置给父盒子，值越大，效果越不明显，看起来近大远小

3、backface-visibility

设置元素背面是否可见

十、动画

1、定义动画序列：

a、通过@keyframes指定动画序列名称，主要先定义再调用；

b、通过百分比将动画序列分割成多个节点；（0%-100%、from-to）

c、在各节点中分别定义各属性

d、通过animation将动画应用于相应元素；

2、调用：animation

a、animation-name设置动画名称

b、animation-duration动画持续时间

c、animation-delay动画延时时间

d、animation-timing-function动画执行速度，linear（匀速）、ease等

e、animation-play-state动画播放状态，play、paused（暂停）、running等

f、animation-direction动画播放方向，默认值normal，alternate动画逆播（动画先正向播放再反向播放）

g、animation-fill-mode动画执行完毕后状态，forwards（结束时的状态）backwards（执行之前的状态）等

h、animation-iteration-count动画执行次数，infinite（无限次）等

i、steps(60) 表示动画分成60步完成

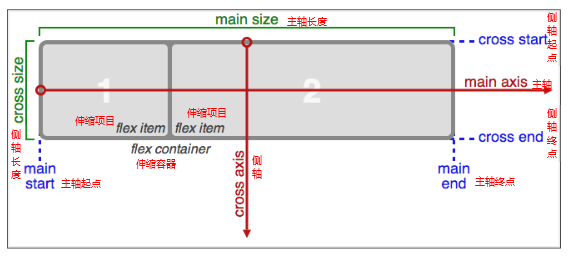
十一、伸缩布局:控制元素对齐方式

主轴：Flex容器的主轴主要用来配置Flex项目，默认是水平方向

侧轴：与主轴垂直的轴称作侧轴，默认是垂直方向的

方向：默认主轴从左向右，侧轴默认从上到下

主轴和侧轴并不是固定不变的，通过flex-direction可以互换。



1、必要元素：

a、指定一个盒子为伸缩盒子 display: flex（父元素）

b、设置属性来调整此盒的子元素的布局方式 例如 flex-direction

c、明确主侧轴及方向

2、各属性详解

a、flex-direction调整主轴方向（默认为水平方向）

row、column、row-reverse行倒序排列、column-reverse列倒序排列

b、justify-content调整主轴对齐

flex-start、flex-end、center、space-between、 space-around

c、align-items调整侧轴对齐

flex-start、flex-end、center、flex-stretch、space-between、 space-around

d、flex-wrap控制是否换行

nowrap、wrap

e、align-content堆栈（由flex-wrap:wrap产生的独立行）对齐

flex-start 、flex-end、center

space-around 行平均分布、space-between 两端对齐、space-streach 拉伸对齐

f、flex-flow是flex-direction、flex-wrap的简写形式

g、flex控制子元素的缩放比例，分配的是剩余空间（独立属性）

h、order控制子元素的排列顺序

**十二、多列布局**

-webkit-columns-width 定义每列的宽度

-webkit-count 定义列数

-webkit-column-gap 定义两栏之间的间距uii

属性：length 长度 = column-rule-length

style 样式 = column-rule-style

color 颜色 = column-rule-color

-webkit-column-rule 设置列之间的分割线

-webkit-column-span 定义跨列显示 默认1，all 横跨所有列

-webkit-column-fill 定义列的高度是否统一，auto，balance 统一设置成内容最多的那一列的高度

**十三、web字体**

@font-face 指定自定义字体

font-family 给自定义字体起名字

src : url () 引入字体文件

**十四、兼容性**

通过http://caniuse.com/ 可查询CSS3各特性的支持程度，一般兼容性处理的常见方法是为属性添加私有前缀，如不能解决，应避免使用，无需刻意去处理CSS3的兼容性问题。

**十五、**

**canvas 标签**

canvas主要属性和方法

|  |  |
| --- | --- |
| 属性 | 描述 |
| width | 设置canvas的宽度 |
| height | 设置canvas的长度 |

|  |  |
| --- | --- |
| 方法 | 描述 |
| save() | 保存当前环境的状态 |
| restore() | 返回之前保存过的路径状态和属性 |
| createEvent() |  |
| getContext() | 返回一个对象，指出访问绘图功能必要的API |
| toDataURL() | 返回canvas图像的URL |

颜色、样式和阴影属性和方法

|  |  |
| --- | --- |
| 属性 | 方法 |
| fillStyle | 设置或返回 填充绘画的颜色、渐变或模式 |
| strokeStyle | 设置或返回 笔触的颜色、渐变或模式 |
| shadowColor | 设置或返回 阴影的颜色 |
| shadowBlur | 设置或返回用于阴影的模糊级别 |
| shadowOffsetX | 设置或返回阴影距形状的水平距离 |
| shadowOffsetY | 设置或返回阴影距形状的垂直距离 |
| 方法 | 描述 |
| createLinearGradient() | 创建线性渐变（用在画布内容上） |
| createPattern() | 在指定的方向上重复指定的元素 |
| createRadialGradient() | 创建放射状/环形的渐变（用在画布内容上） |
| addColorStop() | 规定渐变对象中的颜色和停止位置 |

线条样式属性和方法

|  |  |
| --- | --- |
| 属性 | 描述 |
| lineCap | 设置或返回线条的结束端点样式 |
| lineJoin | 设置或返回两条线相交时，所创建的拐角类型 |
| lineWidth | 设置或返回当前的线条宽度 |
| miterLimit | 设置或返回最大斜接长度 |

矩形方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| rect() | 创建矩形 |
| fillRect()（建议使用这种方式画矩形） | 绘制"被填充"的矩形 |
| strokeRect() | 绘制矩形（无填充） |
| clearRect() | 在给定的矩形内清除指定的像素 |

路径方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| fill() | 填充当前绘图（路径） |
| stroke() | 绘制已定义的路径 |
| beginPath() | 起始一条路径，或重置当前路径 |
| moveTo() | 把路径移动到画布中的指定点，不创建线条 |
| closePath() | 创建从当前点回到起始点的路径 |
| lineTo() | 添加一个新点，创建从该点到最后指定点的线条 |
| clip() | 从原始画布剪切任意形状和尺寸的区域 |
| quadraticCurveTo() | 创建二次贝塞尔曲线 |
| bezierCurveTo() | 创建三次方贝塞尔曲线 |
| arc() | 创建弧/曲线（用于创建圆形或部分圆） |
| arcTo() | 创建两切线之间的弧/曲线 |
| isPointInPath() | 如果指定的点位于当前路径中，返回布尔值 |

转换方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| scale() | 缩放当前绘图至更大或更小 |
| rotate() | 旋转当前绘图 |
| translate() | 重新映射画布上的 (0,0) 位置 |
| transform() | 替换绘图的当前转换矩阵 |
| setTransform() | 将当前转换重置为单位矩阵。然后运行 transform() |

文本属性和方法

|  |  |
| --- | --- |
| 属性 | 描述 |
| font | 设置或返回文本内容的当前字体属性 |
| textAlign | 设置或返回文本内容的当前对齐方式 |
| textBaseline | 设置或返回在绘制文本时使用的当前文本基线 |

|  |  |
| --- | --- |
| 方法 | 描述 |
| fillText() | 在画布上绘制"被填充的"文本 |
| strokeText() | 在画布上绘制文本（无填充） |
| measureText() | 返回包含指定文本宽度的对象 |

图像绘制方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| drawImage() | 向画布上绘制图像、画布或视频 chrome不支持 |

像素操作方法和属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| width | 返回 ImageData 对象的宽度 |
| height | 返回 ImageData 对象的高度 |
| data | 返回一个对象，其包含指定的 ImageData 对象的图像数据 |

|  |  |
| --- | --- |
| 方法 | 描述 |
| createImageData() | 创建新的、空白的 ImageData 对象 |
| getImageData() | 返回 ImageData 对象，该对象为画布上指定的矩形复制像素数据 |
| putImageData() | 把图像数据（从指定的 ImageData 对象）放回画布上 |

图像合成属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| globalAlpha | 设置或返回绘图的当前 alpha 或透明值 |
| globalCompositeOperation | 设置或返回新图像如何绘制到已有的图像上 |

其他方法

|  |  |
| --- | --- |
| 属性 | 描述 |
| globalAlpha | 设置或返回绘图的当前 alpha 或透明值 |
| globalCompositeOperation | 设置或返回新图像如何绘制到已有的图像上 |

智能表单的使用和规范

|  |  |
| --- | --- |
| Input表单的type新属性值 |  |
| Type="email" | 限制用户输入必须为Email类型 |
| Type="url" | 限制用户输入必须为URL类型 |
| Type="date" | 限制用户输入必须为日期类型 |
| Type="time" | 限制用户输入必须为时间类型O |
| Type="month" | 限制用户输入必须为月类型O |
| Type="week" | 限制用户输入必须为周类型O |
| Type="number" | 限制用户输入必须为数字类型 |
| Type="range" | 产生一个滑动条的表单 |
| Type="search" | 产生一个搜索意义的表单 配合results="n"属性 C |
| Type="color" | 生成一个颜色选择表单 |

新增的表单属性

|  |  |  |
| --- | --- | --- |
| 属性 | 值 | 描述 |
| required | required | 表单拥有该属性表示其内容不能为空，必填 |
| placeholder | 提示文本 | 表单的提示信息，存在默认值将不显 |
| autofocus | autofocus | 自动聚焦属性，页面加载完成自动聚焦到指定表单 |
| Pattern | 正则表达式 | 输入的内容必须匹配到指定正则 |

css3新增的文本属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| text-overflow | 设置或检索是否使用一个省略标记（...）标示对象内文本的溢出 |
| text-align | 设置或检索对象中文本的对齐方式 |
| text-transform | 检索或设置对象中的文本的大小写 |
| text-decoration | 复合属性。检索或设置对象中的文本的装饰，如下划线、闪烁等 |
| text-decoration-line | 检索或设置对象中的文本装饰线条的位置。 |
| text-decoration-color | 检索或设置对象中的文本装饰线条的颜色。 |
| text-decoration-style | 检索或设置对象中的文本装饰线条的形状。 |
| text-shadow | 设置或检索对象中文本的文字是否有阴影及模糊效果 |
| text-fill-color | 设置或检索对象中的文字填充颜色 |
| text-stroke | 复合属性。设置或检索对象中的文字的描边 |
| text-stroke-width | 设置或检索对象中的文字的描边厚度 |
| text-stroke-color | 设置或检索对象中的文字的描边颜色 |
| tab-size | 检索或设置对象中的制表符的长度 |
| word-wrap | 设置或检索当当前行超过指定容器的边界时是否断开转行 |

text-overflow属性

作用：设定内容溢出状态下的文本处理方式。

取值：

clip： 默认值

当对象内文本溢出时不显示省略标记（...），而是将 溢出的部分裁切掉。

ellipsis：

当对象内文本溢出时显示省略标记（...）。

注意:该属性需要和over-flow:hidden属性(超出处理)还有white-space:nowrap(禁止换行)配合使用，否则无法看到效果

text-align属性

作用:设定文本对齐方式

取值：

left ：默认值 内容左对齐。

center：内容居中对齐。

right： 内容右对齐。

justify： 内容两端对齐。写本文档时仅Firefox能看到正确效果

start： 内容对齐开始边界。（CSS3）

end： 内容对齐结束边界。（CSS3）

text-transform属性

作用：设定文本的大小写等形式的转换

取值：

none： 默认值 无转换

capitalize： 将每个单词的第一个字母转换成大写

uppercase： 转换成大写

lowercase： 转换成小写

full-width： 将左右字符设为全角形式（CSS3）不支持

full-size-kana：将所有小假名字符转换为普通假名（CSS3）不支持

例如：土耳其语。

text-decoration属性

作用：设定文本修饰线。

取值：

[ text-decoration-line ]：不支持

指定文本装饰的种类。相当于CSS1时的text-decoration属性

[ text-decoration-style ]：不支持

指定文本装饰的样式。

[ text-decoration-color]：不支持

指定文本装饰的颜色。

blink： 指定文字的装饰是闪烁。 仅opera和firefox

例如:text-decoration : overline CSS1实例

text-decoration : #F00 double overline CSS3实例

备注：目前主要浏览器都没有实现上述属性，但是依然可以使用CSS1的实例方式

text-decoration-line属性, 备注:目前大部分浏览器未实现该属性。

作用：设定文本修饰线的位置。

取值：

none：默认值

指定文字无装饰

underline：

指定文字的装饰是下划线

overline：

指定文字的装饰是上划线

line-through：

指定文字的装饰是贯穿线

text-decoration-style属性, 备注:目前大部分浏览器未实现该属性。

作用：设定文本修饰线的样式。

取值：

solid：默认值、实线

double： 双线

dotted： 点状线条

dashed： 虚线

wavy： 波浪线

text-shadow属性

作用:设定文本的阴影效果

取值：

none：默认值，无阴影

text-stroke属性

作用:文本边框颜色，指定文字描边部分的颜色

取值:

[ text-stroke-width ]：

设置或检索对象中的文字的描边厚度

[ text-stroke-color ]：

设置或检索对象中的文字的描边颜色

tab-size属性

作用:设定一个tab缩进键，在页面中的显示长度。

取值:　默认值：８

长度或者整数值

整数值 : z-index:1 此处的1就是整数值，不需要单位，类似倍数。

长度: margin:10px 此处的10px是长度值。  
 注意:该属性只在<pre>标签之内(预格式化状态)显示才会有效。因为浏览器会自动忽略空白字符。

opera和火狐浏览器需要使用浏览器私有前缀。

word-wrap属性

作用:溢出文本(特指类英文文字)的处理方式。

取值:

normal： 默认值

允许内容顶开或溢出指定的容器边界。

break-word：

内容将在边界内换行。如果需要，单词内部允许断行。

移动web开发重难点

第一天：

什么是适配

适配的解决方案（重点）

什么是流式布局

\*什么是viewport（重点）

viewport各个参数的含义（重点）

如何构建一个标准的移动web开发页面

为什么所有的盒子都要以边框开始计算

如何清除移动端高亮效果

Input清除默认的样式

为什么要对页面进行最小宽度和最大宽度的限制

搜索按钮调用

同类型选择器的使用

第二天

\*熟悉touch事件下面的三个常用事件（重点）

了解移动端轮播图制作的思路

满屏页面的设置窗体顶端

文本环绕，右侧内容占满的设置

基本完成京东产品分类页面的布局

第三天

第四天

第五天

第六天

jquery获取触摸点的方法

underscore模板渲染页面的三个步骤

Bootstrap基本组件的使用

Bootstrap轮播图提供的切换上一张下一张方法

覆盖Bootstrap的默认样式

使用伪元素制作半圆效果

根据Bootstrap手册完成分类区域

根据Bootstrap手册完成tab栏

第七天

函数是重复利用的代码块

js中的任何一个全局函数或变量都是window的属性

**移动web开发**

**第一天**

**一、适配问题解决方案：流式布局 + viewport**

1、流式布局

百分比布局，通过设置盒子的宽度为百分比来根据屏幕的大小进行伸缩，特点是不受固定像素的限制，内容向两侧填充

2、viewport

在移动端用来承载网页的这个区域，就是我们的视觉窗口，也叫viewport（视口）, 这个区域可设置高度宽度，可按比例放大缩小，而且能设置是否允许用户自行缩放。

<meta name = "viewport">让浏览器知道设置了viewport，viewport只在移动端识别

3、viewport 参数

width：定义viewport的宽度，有一个特殊值 device-width当前设备的宽度

initial-scale：默认的初始缩放比

设置1.0，和PC端页面的显示比例一致，一比一显示在移动端

user-scalable：是否允许用户自行缩放 1-yes，0-no

maximum-scale：最大缩放比

minimum-scale：最小缩放比

4、构建标准的移动端web开发页面

<meta name = "viewport" content = "width = device-width,initial-scale = 1.0, user-scalable = 0">

快捷键—meta：vp

**二、在写项目要注意的问题**

base样式定义：reset css 重置默认的、浏览器自带的一些样式（所有的标签和伪类），目的是保持各种终端显示一致

1、在移动端特殊的设置

①清楚点击高亮效果

-webkit-tap-highlight-color: transparent;

②设置所有的盒子的宽度以边框开始计算

在移动端通常使用的是百分比布局，那么这样的布局如果使用border或者padding会使容器的宽度超出屏幕的宽度产生滚动条。

-webkit-box-sizing: border-box;

box-sizing: border-box;

③清除input默认的样式

- webkit-appearance: none;

2、最小宽度、最大宽度的限制

min-width：300px; //适配小屏幕设备，为了有较好的布局效果。

max-width：640px;//保证页面在尺寸比较大的设备当中保证页面的效果也就是清晰度

3、物理像素和图标模糊度问题

解决方案，采用压缩图标尺寸的方式来解决。

如果是Img使用直接设置宽高的方式来压缩。

如果是背景使用的是设置background-size的方式来压缩

4、搜索按钮调用

在移动端调用输入法，弹出的小键盘enter键会变成搜索按钮

<form action = "#">

<input type = "search" placeholder = "提示" />

</form>

5、结构性伪类原则器---同类型选择器

E:first-of-type匹配同类型中的第一个元素E。

E:last-of-type匹配同类型中的最后一个元素E。

E:nth-of-type(n) 匹配同类型中的第n个元素E。

**第二天**

**一、Touch事件**

**touchstart：当手指触碰到屏幕的时候触发**

绑定事件的方法：dom.addEventListener("touchstart", function(e){});

事件返回的e对象包含那些移动端特有的属性：

targetTouches 目标元素的所有当前触摸

changedTouches 页面上最新更改的所有触摸

touches 页面上的所有触摸

**touchmove：当手指在屏幕上滑动时连续触发**

绑定事件的方法：dom.addEventListener("touchmove", function(e){});  
 事件返回的e对象包含那些移动端特有的属性：

originalEvent  是jquery 封装的事件。

targetTouches 目标元素的所有当前触摸

changedTouches 页面上最新更改的所有触摸

touches 页面上的所有触摸

在滑动的时候不断给图片盒子做定位，来达到滑动的效果

定位的位置，当前的图片的定位加上移动的距离

在开始滑动的时候就要清楚定时器，move的时候清除过渡，-index\*width + distanceX

**touchend：当手指在屏幕上结束滑动时触发**

绑定事件的方法：dom.addEventListener("touchend", function(e){});

事件返回的e对象包含那些移动端特有的属性：

changedTouches 页面上最新更改的所有触摸

touchcancel：系统停止跟踪触摸时候会触发，系统被动终止，例如alert。

注意：在touchend事件的时候event只会记录changedtouches

clientX:触摸目标在视口中的X坐标。

clientY:触摸目标在视口中的Y坐标。

pageX：触摸目标在页面中的x坐标。

pageY：触摸目标在页面中的y坐标。

screenX:触摸目标在屏幕中的x坐标。

screenY:触摸目标在屏幕中的y坐标。

**二、过渡和动画结束时间**

1、transitionEnd：过渡结束后触发。

绑定事件的方法：

dom.addEventListener("webkitTransitionEnd", function(e){});

dom.addEventListener("transitionEnd", function(e){});

2、animationEnd：动画结束后触发。

绑定事件的方法：

dom.addEventListener("webkitAnimationEnd", function(e){});

dom.addEventListener("animationEnd", function(e){});

3、gesture事件

gesturestart：当一个手指触摸屏幕之后，第二个手指再触摸屏幕时触发。

gesturechange：当上面的事件触发后立即触发。

gestureend：第二根手指离开屏幕时触发，之后将不会再次触发gesturechange。

在event当中会返回另外两个参数

scale 根据两个手指的滑动距离计算的缩放比例 初始1

rotation根据两个手指的滑动距离计算的旋转角度 初始 0

4、全屏单页布局： html,body{height:100%}

**第四天**

**一、响应式开发**

1、响应式布局， 就是一个网站能够兼容多个终端。

2、响应式开发的原理

CSS3中的Media Query（媒介查询，and之后一定要加空格@media screen and (min-width: 768px) and (max-width: 992px) {}; 通过查询screen的宽度来指定某个宽度区间的网页布局。

超小屏幕（移动设备）768px以下

小屏设备 768px-992px

中等屏幕 992px-1200px

大屏设备 1200px以上

3、响应式开发和移动web开发的区别

|  |  |  |
| --- | --- | --- |
| 开发方式 | 移动web开发+PC开发 | 响应式开发 |
| 应用场景 | 一般在已经有PC段的网站，开发移动站的时候，只需单独开发移动端 | 针对新建站的一些网站，现在要求适配移动端，所以就一套页面兼容各种终端，灵活 |
| 开发 | 针对性强，开发效率高 | 兼容各种终端，效率低 |
| 适配 | 只是陪移动设备，pad上体验相对较差 | 可以适配各种终端 |
| 效率 | 代码简洁，加载快 | 代码相对复杂，加载慢 |

bootStrap

bootStrap框架----当前最流行的前端UI框架（有预制界面组件）

特点：组件简洁大方，代码规范精简，界面自定义性强

优点：

有自己的生态圈，不断的更新迭代

提供了一套简洁、直观、强悍的组件

标配准化的html+css编码规范

让开发更简单，提高了开发的效率。

3、布局容器

.container固定宽度并且支持响应式布局的容器

默认margin:55px; padding:15px;

.container-fluid类用于 100% 宽度，占据全部视口（viewport）的容器。

4、删格系统

行：row 通过自身的拉伸来填充父容器的内边距

删格参数：

|  | 超小屏幕 手机(<768px) | 小屏幕 平板(≥768px) | 中等屏幕(≥992px) | 大屏幕 (≥1200px) |
| --- | --- | --- | --- | --- |
| 栅格系统行为 | 总是水平排列 | 开始是堆叠在一起的，当大于这些阈值时将变为水平排列C | | |
| .container最大宽度 | None（自动） | 750px | 970px | 1170px |
| 类前缀 | .col-xs-\* | .col-sm-\* | .col-md-\* | .col-lg-\* |
| 列数(column) | 12 | | | |
| 可嵌套 | 是 | | | |
| 偏移（Offsets） | .col-xs-offset-\* | | | |
| 可见、隐藏 | .visible-xs-\* .hidden-xs | | | |

**第五天 微金所**

**一、媒体查询**

在响应式开发中使用媒体查询中的screen查询浏览器的宽度来定义不同宽度区间的样式和布局。

代码：

@media screen and (min-width: 768px){}

@media screen and (min-width: 992px){}

@media screen and (min-width: 1200px){}

或者

@media (max-width: 767px){}

@media (min-width: 768px) and (max-width: 991px){}

@media (min-width: 992px) and (max-width: 1199px){}

@media (min-width: 1200px){}

二、结构选择器

div a{}

div > a {}

div + div {} 选择目标元素的下一个元素

div ~ div {} 选择目标元素之后的所有同级元素

三、Bootstrap常用样式

1、container类

用于定义一个固定宽度且居中的版心

2、row类

每一个列默认有一个15px的左右外边距，row类的一个作用就是通过左右-15px屏蔽掉它

xs : 超小屏幕 手机 (<768px)

sm : 小屏幕 平板 (≥768px)

md : 中等屏幕 桌面显示器 (≥992px)

lg : 大屏幕 大桌面显示器 (≥1200px)

3、hidden 类

hidden-xs,hidden-sm,hidden-md,hidden-lg在不同的屏幕下隐藏。

4、text-\* 类

text-center 文本居中

text-left 文本左对齐

text-right 文本右对齐

5、pull-\* 类

pull-left 左浮动类

pull-right 右浮动类

6、center-block 类

让一个固定宽度的元素居中。

7、display: table-cell

四、rem

相对长度单位，相对于html根元素font-size计算值得倍数

test布局 //白青衣/rem插件/rem/rem.css

flexible布局//白青衣/rem插件/flexible/rem.css

淘宝这个方案可以在任意设计稿尺寸下使用（地址：https://github.com/amfe/lib-flexible）

1）除font-size外，

其它大小都根据750标注稿的尺寸，转换成rem单位的值，转换方法为：

标注稿尺寸 / 标注稿基准字体大小；

2）标注稿基准字体大小 = 标注稿宽度 / 10，

如标注稿宽为750，标注稿基准字体大小为75；标注稿宽为640，标注稿基准字体大小为64；

em的大小是根据父元素的font-size来设置的

rem的大小是根据html标签的font-size来设置的

标签属性

提供给屏幕阅读器的属性，我们可以忽略

role、 aria-\*、 class="sr-only"

指定插件类型，和控制的目标元素

data-toggle

data-target

a标签也可以通过href来指定被控制的目标元素

轮播图

1、自适应图片：

在pc端使用的是背景图片，在移动端使用的是图片

2、在写静态demo的时候使用的是响应式工具来完成的图片响应，但是会加载两种图片那么这时候就需要做图片的响应式了，首先我们准备了json 中有两种数据，然后ajax请求 到数据做缓存 如果请求过就直接取缓存的，然后通过模版引擎解析成html，最终把解析完成的html渲染在页面当中。

**七、自定义字体：声明自定义字体的名字，引入字体文件，指定字体文件的格式**

@font-face{

font-family: "wjs";

src: url(../fonts/MiFie-Web-Font.eot) format("embedded-opentype");

src: url(../fonts/MiFie-Web-Font.svg) format("svg");

src: url(../fonts/MiFie-Web-Font.ttf) format("truetype");

src: url(../fonts/MiFie-Web-Font.woff) format("woff");

}

wjs\_icon{font-family: wjs; }

通过伪类来定义一个字体图标，字体图标设计出来的时候一般都会有文档参考

wjs\_icon\_phone:before{  
 content: "\e908";

}

**后台**

准备模板对象

var tempaltePoint = \_.template($('#templata\_point').html());

传入数据

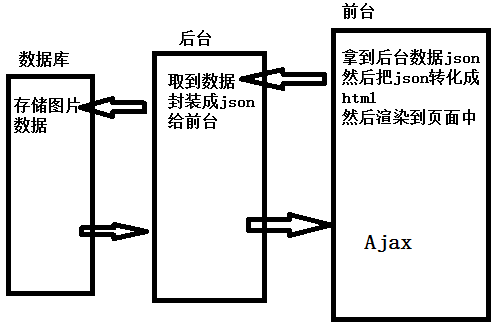
var abc = {model: data};

var pointHtml = tempaltePoint(abc);

传入数据后，这时候已经将模板解析成HTML字符，最后添加到页面结构中

$('.carousel-indicators').html(pointHtml);

前后台交互过程



**第六天 less**

**一、简介**

1、什么是less[中文网](<http://lesscss.cn/>)

css的预处理语言，LESS 包含一套自定义的语法及一个解析器，用户根据这些语法定义自己的样式规则，这些规则最终会通过解析器，编译生成对应的 CSS 文件。

2、使用less

less文件只有在被编译后才能够被浏览器识别并使用

3、客户端调用方式在.less中@charset "utf-8";

①link引用.less文件 type = "type/less",引用less.js插件,不建议使用

l ess.js 解析less文件动态加载head，异步加载less文件

②直接引用less.css

/\*\*/会编译在css文件当中

//不会编译在css中

**二、语法**

1、变量

LESS 允许开发者自定义变量，变量可以在全局样式中使用，使得样式修改起来更加简单。

@mainColor: #E93223;

body{color: @mainColor}

2、Mixin混入

Mixin（混入）特性，它是多重继承的一种实现，在 LESS 中，混入是指在一个 CLASS 中引入另外一个已经定义的 CLASS，就像在当前 CLASS 中增加一个属性一样。

.red{color: @mainColor};

.border{border: 1px solid #ccc}

////方法混合.redBorder(){color: @mainColor;border: 1px solid #ccc}

样式混合（通过class）：.mixin-class{.red();.border();}

3、嵌套

在我们书写标准 CSS 的时候，遇到多层的元素嵌套这种情况时，我们要么采用从外到内的 选择器嵌套定义，要么采用给特定元素加 CLASS 或 ID 的方式

#wjs\_banner{

>div.item{

img\_box{  
 redBox();

&:hover{

color:@mainColor;

}

}

}

}

4、导入Import

@import "base";//引入其他的less文件

.f\_left{float: @right}

5、运算及内置函数

在我们的 CSS 中充斥着大量的数值型的 value，比如 color、padding、margin 等，这些数值之间在某些情况下是有着一定关系的，那么我们怎样利用 LESS 来组织我们这些数值之间的关系呢？

<http://www.1024i.com/demo/less/reference.html>

运算：

@back:#333;

.test{

border: 1px solid @back\*2

}

内置函数：

desaturate(@color, 10%);

lighten(@color, 10%);

darken(@color, 10%);

fadein(@color, 10%);

fade(@color, 10%);

**面向对象**

**第一天**

面向对象

面向对象的基本概念

js 是一个基于对象的多范式的编程语言

多范式: 编程风格

面向过程的方式编程

面向对象的方式编程

函数式 ...

-> 函数式的一个特点: 递归与链式

jQuery 就是典型的链式编程风格

$( 'div' ).css( 'border', '1px solid red' )

.css( 'width', '400px' )

什么是面向对象

1、面向对象的概念

面向：将脸朝向......-> 关注，用

面向过程：关注编程步骤，细节的编程方式（自己动手完成）

面向对象：关注对象，找到对象，让它帮你做，等结果

面向过程就是亲力亲为, 事无巨细, 面面俱到, 步步紧跟, 有条不紊

面向对象就是找一个对象, 指挥得结果

面向对象不是面向过程的替代, 而是面向过程的封装

2、对象是什么

A.对象是存储数据的数据集，是提供功能的功能集，对象可以用来保存数据

B.使用对象来添加一些特定的功能

var itcast = { tag:function(){}, addCss:function(){}}

使用方法： itcast.tag(参数)

3、在js中，对象就是键值对的集合，名词

4、使用面向对象编程（实际开发的时候如果需要完成一个功能）

①首先考虑系统是否提供了对象（使用现有对象）

document, 标签对象

②如果系统没有提供，就自己创建一个对象或者第三方（自定义对象）

注意：函数==>使用js代码编写的一个普通的function就是函数

方法==>使用对象添加的功能（函数）就是方法

三、面向对象的特性

1、抽象性：抽取我们所需要的数据信息

2、封装性：留下一些能使用的特性

3、继承性

四、值类型与引用类型

数字 + 上下文 = 信息

内存就是一个可以存储数字（数据）的盒子

1、JavaScript 中的数据类型

A基本类型（值类型）: 数字, 布尔, 字符串

B复合类型:（引用类型 : 对象( 数组, 时间, 正则, Object , 函数等 )

C空类型: Undefined, Null

复合类型除函数其它的无法通过typeof获得数据类型，要使用Object.prototype.toString.apply()

2、数据存储模型:“盒子” 与数据的逻辑结构

A.基本数据类型和空类型的存储模型：一个方格里面放一个数据

B.复合类型的存储模型：

复合类型的对象是一个单独的内存区域, 对象有什么属性, 那么内存区域中就有什么数据。变量 p 只存储该对象的 '地址'. 所以 p 不是真正存储数据的区域.

3、值类型与引用类型的存储特征

①值类型的数据, 只需要开辟一段内存存储数据即可

var a = 123; var b = 'abc'; var c = true;

②对象类型( 引用类型 ). 对象才是真正的数据, 需要占据单独的内存.

而变量名只是存储着对象的内存地址( 引用 ).

即创建一个对象并赋值, 实际上需要两块内存空间. 一个存储数据( 对象 ),

另一个存储变量名以引用对象.

4、值类型与引用类型的赋值与传参的特性

-> 赋值: 将原变量中的数据拷贝一份, 然后赋值给新的变量中

① 值类型//拷贝的是实际的数据

② 引用类型//把变量的引用地址拷贝一份给新的变量，此时新的变量旧的变量都是指向 了同一个对象

var o1 = { num: 123 };

var o2;

// 赋值

o2 = o1; // 将 o1 中存储的内容 拷贝一份 存储到 o2 中

// o1 中存储的是引用, 或 '地址'

o1 和 o2 虽然是不相同的两个变量, 即两个独立的盒子. 但是由于存储的地址相同.

在访问数据的时候, o1 与 o2 也是访问同一个数据, o1 将数据修改了, o2 再读取,

读取的就是修改后的数据.

-> 函数参数传递

-> 什么是函数参数传递

在调用函数的时候, 会将参数中表示的数据拷贝一份, 然后给参数赋值

function foo( num ) {}

// 调用

var a = 123;

foo( a ); // 调用函数就是要执行函数// 将 a 中存储的数据 拷贝一份

// 进入 函数 foo// 给参数赋值, 相当于 num = a

// 进入函数体, 开始执行函数

总结：值类型引用类型的数据的传参

参数是值类型，函数中对参数的任何改变都不会影响到函数

参数是引用类型，函数中对参数对象的改变会影响到函数外的实参

14. 深拷贝与浅拷贝

-> 什么是拷贝: 就是创建一个与目标数据一模一样的数据

-> 案例:

var p = { name: '张三' };

var p1 = p; // 是否存在拷贝

// 一般描述拷贝是指拷贝对象

p1 = {};

p1.name = p.name;

// 才是拷贝

-> 案例, 给 对象 p 提供一个 clone 方法, 完成拷贝

-> 案例:

有一辆汽车 Car: name=保时捷

有一个人 Person : name=张三

-> 如果对象的属性也是一个引用类型, 拷贝的时候不重新创建一个新的对象来实现该属性的拷贝, 那么就是浅拷贝.

即任何不完全的拷贝都是浅拷贝

-> 将两个对象完全从内存中隔离开, 就是深拷贝. 即每一个引用属性, 以及引用属性的引用属性, ... 全部拷贝出来.

五、构造函数( 构造器 controctor )的作用

1、js 中对象的动态特性

动态的为对象添加属性

2、方式：点语法与关联数组语法

o.name = 'jim'; // 点语法赋值

console.log( o.name ); // 点语法取值

o[ 'name' ] = 'tom'; // +关联数组语法赋值

console.log( o[ 'name' ] ); // 关联数组语法取值

问题:o[ name ] = 'jack'; // 如果 name 是一个变量, 里面存储的是 字符串, 也是可以的

3、例创建一个 Person 对象

var p = {}; // 什么都没有的对象

p.name = '张三';

p.age = 30;

p.gender = '男';

-> 简化: 提供一个创建 Person 对象的函数

function createPerson( name, age, gender ) {

var p = {};

p.name = name;

p.age = age;

p.gender = gender;

return p;

}

var p1 = createPerson( 'jim', 19, '男' );

var p2 = createPerson( 'lily', 18, '女' );

这个( 这种类型 )的函数就是用来创建对象的, 即生产对象. 常常将这类函数

称为 '工厂函数'

工厂方法：

工厂方法就是一个函数，这个函数就是用来创建并返回一个对象，而且每次创建和返回的对象的特征都是一样的，像流水线下来的一样，这样的“流水线作业生产对象”函数叫做工厂方法。

5、构造方法创建对象：（构造器：用来创建对象的一个函数）

-> 构造器中不需要 return 语句. 一般也可以不写

-> 调用构造器的时候, 使用 new 运算符引导

-> 在构造器中 this 表示当前“new”出来的对象，给对象提供成员使用 this.xxx 的 方式

-> 将 createPerson 改造成构造器

// 构造器的定义、结构

function 构造函数名（参数）{

this.属性 = 属性值；

}

var p = new 构造函数名（）；

function createPerson( name, age, gender ) {

this.name = name;

this.age = age;

this.gender = gender;

}

// 调用构造器创建对象

var p = new createPerson( '李磊', 19, '男' );

-> 构造器创建对象的本质: 还是使用对象的动态特性

-> 首先执行 new 运算符. 即创建对象. 可以简单看做为 {} 的简写

var p = new ... '相当于' var p = {}

-> 调用构造器. 并隐含一个参数, 即刚刚创建的对象.

-> 在构造器中使用 this 引用刚刚创建出来的对象.

-> 构造器结束是 默认返回 this

-> 补充说明

-> 构造器的名字, 一般使用 Pascal 命名规则( 首字母大写的 )

-> 一般为了与其他面向对象的编程语言( C++, C#, Java )保持一致. 称构造函数名为类名

function Person( name, age, gender ) {

this.name = name;

this.age = age;

this.gender = gender;

}

// 调用构造器创建对象

var p = new Person( '李磊', 19, '男' );

17. 异常与捕获

-> 异常

简单的说, 就是代码在执行过程中出现的错误. 并不是代码的语法错误.

-> 一旦出现异常, 其后的代码, 不再执行

-> try-catch 语法

1) try-catch 形态

try {

有可能出现错误的代码

} catch ( e ) {

处理宠物的异常

}

2) try-catch-finally 形态

finally {

不管错误会不会发生，都会执行的代码

}

无论是出现异常，try语法结束的时候都

-> 自定义抛出异常

一般可以封装函数完成特定的功能. 例如 tag 函数

function tag ( tagName ) {

if ( tagName 不是字符串 ) 抛出异常.

return document.getElementsByTagName ( tagName );

}

-> 抛出异常的语法

throw 对象

18. DOM 的核心内容

-> 什么是 DOM, 为什么需要 DOM

<div><div></div><div></div></div>

-> DOM 操作操作的是什么?

-> 访问各亲属节点

-> 增删改查

-> 学会分析 DOM 树

<div>

<p>你好, 我是一个 <span style="color: red">DOM</span> 树的练习</p>

</div>

在 HTML 文件结构中吗所有的内容都是节点(node)对象. 有的是文本节点.

有的是标签( 元素 element )对象. 还有的是属性节点( attribute node )对象.

19. 访问各个亲属节点

-> 节点对象的常用属性

<node>.nodeType

元素(标签): 1,

属性: 2,

文本: 3

<node>.nodeName

元素(标签): 大写的标签名,

属性: 属性名

文本: #text

<node>.nodeValue

元素(标签): null

属性: 属性赋值等号后面的值

文本: 文本字符串

20. 操作 DOM 就是创建元素, 插入元素, 修改元素, 查询元素, 删除元素

-> 增加

// 创建

document.createElement( '元素名' ) 创建元素标签

document.createTextNode( '文本内容' ) 创建文本节点

// 插入

<parent>.appendChild( 子元素 ) 追加到子元素的结尾

<parent>.insertBefore( 新元素, 旧元素 ) 将新元素插入到旧元素的前面

// 简单的办法

<element>.innerHTML = ...

<element>.innerText = ...

// 增加属性

<element>.属性名 = ...

<element>.setAttribute( 属性名, 值 );

<element>.style.xxxx = ...

-> 删除

<parent>.removeChild( 子元素 );

<element>.setAttribute( 属性, '' )

<element>.xxxx = null

-> 修改

-> 修改属性

<element>.xxx = ...

<element>.setAttribute( 属性名, ... )

-> 修改节点

-> 查询

21. 手动创建一个 table 表格, 并且在里面显示数据

var arr = [

{ name: 'jim1', age: 19, gender: '男' },

{ name: 'jim2', age: 18, gender: '男' },

{ name: 'jim3', age: 20, gender: '男' },

{ name: 'jim4', age: 18, gender: '男' }

];

深拷贝与浅拷贝

六、构造函数的作用

对象的动态特性

动态特性

属性访问点语法

关联数组语法

利用动态特性创建对象

基本步骤

工厂方法简介

构造函数的本质

new 与 构造函数 的执行步骤

图解

第七天

1. 为什么需要原型：减少内存的浪费，提高效率，使代码更加安全

构造器创建对象的时候, 实际上会有成员重复

如果使用 构造器 this.方法名 = function .... 方式创建对象. 那么每一个对象

对应的方法就会重复.

解决办法就是让这个方法( 函数 )共享

①将函数写到外面, 那么 Person 在初始化对象的时候就不会再创建一个函数了.

只需要将 外面的函数引用 交给对象即可.

缺点: 一个对象可能有 n 多方法. 如果将所有的东西 都放到外面, 与其他库

冲突的几率就会变大. 所以不宜采取该方法.

②将所有的方法( 函数 )都绑定到一个对象中.

③js 原生就支持解决该问题的办法：使用prototype属性

每一个函数都有 一个属性 prototype，该属性指向一对象（Fun.prototype）.

(重点) 每一个由该函数作为构造器创建的对象, 都会默认连接到该对象上.

如果访问对象的方法, 而对象中没有定义, 就会在这个构造函数.prototype

表示的对象中去找.

-> prototype 就是原型之意

原型就是被克隆的人，构造函数是克隆的机器，实例对象是克隆体

2. 原型就是 构造函数的 prototype 属性, 常常将其称为 原型属性.

原型就是 实例对象的 原型对象，实例对象是由原型对象复制而来的一个对象

例:

function Person () {} // 有了构造函数, 和 原型

var p = new Person(); // 有了实例

3. 一般如何使用原型对象

-> 简单的说就是将共享的方法放到原型中, 而独有数据与行为放在当前对象里

-> 例:

Person( name, age, gender, sayHello, eat, run )

-> 通过动态特性接给原型对象添加成员:这种方式添加的话，原型依然会有constructor的属性，并且指向原来的构造函数

-> 通过Person.prototype指向一个新对象替换旧原型对象：原型里面没有constructor属性，需要手动去添加这个属性（constructor:函数名） 属性, 表示对应的构造函数 )

constructor：构造器，原型的

原型属性--函数

原型对象--实例对象

4. \_\_proto\_\_：用来访问原型的一个属性

-> 访问

使用构造函数, 就使用 prototype 属性访问原型

使用实例对象, 就使用非标准的 \_\_proto\_\_ 属性访问原型(对象名.constructor.prototype)

5. 继承

-> 什么是继承

自己没有, 别人有, 拿过来自己用, 就好像自己的一样.

-> 原型与实例对象

在 js 中, 方法定义在原型对象中, 而属性定义在实例对象中

调用方法的时候, 实例对象本身是没有该成员的, 但是依旧可以调用

该方法, 好像这个方法就是该实例对象的一样. 因此, 我们称该实例对象

继承自 原型对象

-> 任何一个实例, 都是继承自其原型对象的. 即原型式继承.

6. 为什么需要继承

-> 编程的发展

复用( 重复使用 )

div 标签对象 nodeName, nodeType, nodeName, ...

appendChild, insertBefore, getElementsByTagName, ...

a 标签对象

baseElement

-> js 运行效率

共享特性

复用

9. 属性访问原则( 重点 )

1) 对象在调用方法或访问属性的时候, 首先在当前对象中查询. 如果有该成员使用并停止查找.

2) 如果没有该成员就在其原型对象中查找. 如果有该成员即使用, 并停止查找.

3) 如果还没有就到 该对象的 原型对象 的 原型对象中查找.

...

4) 最后会查到 Object.prototype 上. 如果还没有即 返回 undefined.

10. 如果修改原型对象中的属性值会怎样

给当前对象的原型提供的属性赋值, 实际上是给当前对象添加了该属性的新成员

并不会修改运行对象中的成员.

11. 混入( mix ): 将一个对象中的成员加到另一个对象中

var o1 = { name: '张三' };

var o2 = { age: 19 };

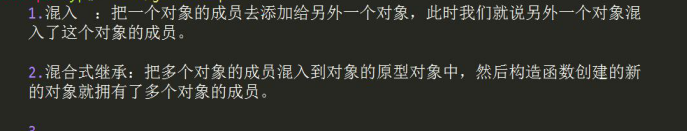
-----------

o2.name = o1.name

利用这个简单的赋值, 就可以

混入使得 o2 具有了 age 和 o1 中的 name. 即将 o1 混入到 o2 中

混入也是一种继承的方式



12. 如何使用代码实现混入

// 考虑需要执行, 写函数即可

// 由于将 一个 对象混入到 另一个对象. 所以有两个参数

私有

function \_\_mix\_\_ ( obj(原对象), obj1(新对象) ) {//将obj1的成员加入到obj中去

// 如何获得 obj1 中的所有成员?

for ( var k in obj1 ) {

obj[ k ] = obj1[ k ];

}

}

在函数中有一个默认的对象arguments存在，它里面存储着所有传入的参数

我的extend方法可以带多个参数，如果是多个参数，要求将所有对象的成员都加到this当前对象上

argument对象：这个对象在每一个函数里都有，代表的就是函数的所有参数，是一个数组

var o = {

extend: function(obj){

for(var i = 0; i < arguments.length; i++){

for (var k in arguments[i]){

this[k] = arguments[i][k];

}

}

}

}

13. 原型式继承

-> 写一个构造函数, 如果需要将其实例继承自某个特定的对象 o. 那么

只需要设置该构造函数的 prototype 属性为 o 即可

function Person (){}

var o = { ... }

Person.prototype = o; // 继承

14. 混合式继承

-> 将多个对象的各个功能混合到一起, 加到构造函数的原型对象上，这些功能是从原型继承而来，所以叫做混合式继承

步骤：

A：给原型添加一个extend方法

function 构造函数 (){}

构造函数.prototype.extend = function(obj){

for (var k in obj){

this[k] = obj[k];

}

};

B.调用extend方法去混入多个对象的功能

构造函数.prototype.extend{

(多个对象)

方法: function(){}

}; //此时原型中就会混入多个对象的功能了

C.使用构造函数创建对象

var test = new 构造函数();

15. 原型链结构：实例对象 -> 对象的原型对象 -> object的原型对象 ->null 链式结构( 原型链 )

16：新对象 Object.create( 原对象 );

以对象为原型创建一个新的对象，用在只有实例对象的时候

var newObj = Object.create(oldObj)

低版本兼容：先判断有没有，如果没有再加上

function createWithObject (obj) {

if (Object.create) {

return Object.create(obj);

}else{

function F(){}

F.prototype = o;

return new F();

}

}

**第三天**

1. 复习

-> 继承

原型式继承

function Person() {}

Person.prototype = 父对象;

var p = new Person();

p 继承自 父对象

混入 ( mix, extend )

function extend( o1, o2 ) {

for ( var k in o2 ) {

o1[ k ] = o2[ k ];

}

}

注意命名规则:

1> 一般 将一个对象混入另一个对象, 使用 mix

2> 如果是当前对象的方法. 是将另一个对象混入当前对象一般使用 extend

2. 对象的原型链

-> 凡是对象都有原型

-> 构造函数 Person 创建的对象 p 有原型 Person.prototype

-> Person.prototype 的原型对象是Object.prototype

-> 结论

1) Person.prototype 是 实例 p 的原型对象, 使用 \_\_proto\_\_ 也可以访问对象的原型对象

2) Person.prototype 的 原型对象是 Person.prototype.\_\_proto\_\_

3) Person.prototype.\_\_proto\_\_ 里的 constructor 是 Object. 所以

Person.prototype.\_\_proto\_\_ 就是 Object.prototype

4) Object.prototype.\_\_proto\_\_ 是 null. 因此表明 Object.prototype 就是顶级.

-> 链式

p --> Person.prototype( p.\_\_proto\_\_ ) --> Object.prototype --> null

-> 系统内置的原型链

[] --> Array.prototype --> Object.prototype --> null

/./ --> RegExp.prototype --> Object.prototype --> null

... ...

3. 绘制数组的原型结构

var arr = [];

// 等价

var arr = new Array();

arr --> Array.prototype --> Object.prototype --> null

4. 练习:

根据下面代码绘制对象的原型链结构

1) function Person( name, age, gender ) {

this.name = name;

this.age = age;

this.gender = gender;

}

function Student () {}

Student.prototype = new Person( '张三', 19, '男' );

var stu = new Student();

2) function Person() {}

var p1 = new Person();

var p2 = new Person();

5. {} 对象的原型链结构

在 js 中 对象 一般都有字面量

123, '123'

数组: []

正则表达式对象: /./

函数: function () {}

...

对象也有字面量: {}

{} --> Object.prototype --> null

注意: {} 与 new Object() 含义相同

6. 动态函数 Function

-> 动态函数就是在运行的过程中, 将一段字符串作为代码运行.

由于字符串可以随意的拼接. 因此得到动态的执行.

-> 定义动态函数, 并执行

-> 使用 Function 构造函数, 创建函数.

Function 是一个构造函数. new Function 得到 一个函数

-> 语法

var demo = prompt("");

var demo2 = new Function( arg0, arg1, ..., argN, body )

demo2();

Function 的所有的参数, 除了最后一个以外, 都是生成的函数的参数

最后一个参数是 函数体

7. 函数的相关的一些参数

①arguments对象：参数对象，保存了调用函数时传入的所有参数，使用数组的索引访问参数.

例如: 写一个函数, 在参数中写任意个参数, 最后求其和

function sum () {

// 所有的参数都会存储到 arguments 中

var sum = 0;

for ( var i = 0; i < arguments.length; i++ ) {

sum += arguments[ i ];

}

return sum;

}

使用arguments改写extend函数

函数名.prototype.extend = function(){

for(var i = 0; i<arguments.length; i++{

for(var k in arguments[i]){

this[k] = arguments[i][k];

}

}

}

②函数名.length, 即函数的 length 属性. 表示 定义函数时, 参数的个数

如果定义函数的时候, 定义了参数. 但是调用的时候又没有传递该参数. 那么该参数在函数内就是 undefined

③函数.name 返回的是函数名

8. 函数的引用 callee 与 caller

js 中函数也是一个对象

-> callee被调用者 在函数的内部, 实现函数递归的时候，一般使用callee表示 当前函数 的引用，

function fn(){

arguments.callee()

}

fn();

-> caller 表示调用函数，调用者

9. eval 函数（立即执行函数，自调用函数）通过ajax获取请求，获得数据

eval 函数与 Function 功能类似. eval 可以直接将字符串作为代码来执行.

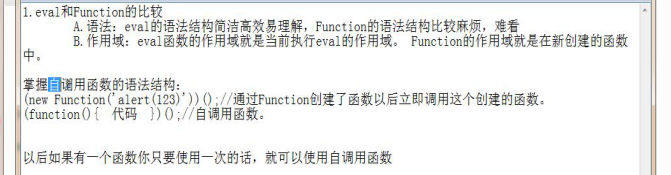
语法:

eval( 语句字符串 )，在eval函数中，使用字符串声明的变量，在Evelyn函数外面可以立即执行

注意, 它好像与当前代码处于同一个作用域

语法上：eval 语法简洁，直接调用即执行

Function创建了一个新的，所有的东西，都只在函数内部起作用，如果要执行，需要调用

除非：1）变量没有声明，直接使用②return

将json字符串变成对象，三种做法

eval做法 var o1= eval（"("+data+")"）;（必须加（）），用来执行jsdaima 的，所有，json字符串中的{}读作了代码快的开始和结束，所有一定要加上（）；

Function 做法var o2 = (new function('return'+data))();

3）var o3 = JSON.parse（dara）; 低版本的浏览器可能不兼容，要求json字符串是严格格式字符串（键必须用引号引起来）

11. 在 js 中 函数 是 Function 的实例

function Person() {}

var p = new Person();

p 是 构造函数 Person 的实例

在 该角度去看, 函数就是对象, Function 就是构造函数

得到 构造-实例-原型 三角形

函数是一个函数对象，认为是一个实例对策function Person(){}

Function是函数的构造函数，，由Function创建函数实例对象，Function是自己的实例，任意一个函数是它的对象，是函数对象创建了自己

相当于：Person =



function Person() {} 为例

12. 给定一个任意的对象, 将其原型链包括对象和函数的结构完整的分析处理

13. instanceof 运算符

-> a instanceof b -> 判断b是否在a的原型链上

-> instanceof ?

-> 错觉: 判断某一个对象是否为某一个构造函数所创建出来的

语法：

对象 instanceof 构造函数（结果是布尔值）

判断构造函数的原型属性是否在对象的原型链上

14. 通过原型链的学习, 可以重新定义 js 的继承

js 的继承: 就是利用对象的动态特性添加成员, 或直接替换对象的方式修改原型链结构. 使得当前对象的原型链上的对象具有某些成员. 那么我的当前对象就可以使用这些成员了.

p -> Person.prototype -> Object.prototype -> null

p -> Person.prototype -> {} -> Object.prototype -> null

过多的依赖原型链继承, 会损耗 性能

如果必须使用原型链继承, 最好提供一些快速访问的方法

1. 复习

加入我希望所有的函数都具有 inherit 功能

function Person() {}

var Student = Person.inherit( 成员 );

// 生成函数 Student, 但是 Student 里面的成员( this.XXX )

// 由参数来提供, 而 Student 的对象, 继承自 Person 的对象

如果希望所有的函数都有 inherit 方法, 那么可以给 Function.prototype 添加该方法

2. 代码的预解析

-> 预解析：在代码执行之前先把代码解释分析一遍，处理一些核心问题，为了提高效率

提前的翻译解释, 在运行代码之前的一个解释.

-> 为什么需要它

-> 编译型语言: C, C++, C#, Java

就是需要一个 "翻译" 程序, 将源代码翻译成计算机可以读懂的二进制数据( 指令 ).

然后存储成可执行文件.

提前翻译好, 运行时直接执行得结果

-> 解释型( 脚本型 ): JavaScript, SQL, ...

代码在执行的时候, 有一个翻译程序, 读一句代码执行一句代码. 再读一句代码,

再执行一句代码.

一句一句的翻译执行. 每次运行都需要翻译一次.

-> 代码在执行之前, 需要快速的 "预览" 一遍. 那么可以尽可能提高执行效率.

3. 在 js 中预解析的特点

-> 代码是如何执行的: 读取 js 文本, 预解析, 一句一句地执行

-> js 在预解析的过程中完成了声明部分的标记与变量作用域的设定

4. 什么是 js 中的声明

-> 简单的说就是让 js 执行引擎知道有什么东西( 标识符 )

console.log( num ); // error: num is not defined

num(); // error: is not function

即代码在执行之前的预解析, 首先让 js 的执行引擎知道在当前运行环境中

有什么东西( 名字, 标识符 )是可以被使用的. 它是变量, 还是函数等?

-> 在 js 中有哪些声明:

1) 标识符的声明( 变量的声明 )

2) 函数的声明

-> 变量的声明:

语法: var 变量名;

目的: 告诉解释器, 有一个名字是一个变量, 在当前环境中可以被使用.

语句: 就是可以执行的东西.

var a = 123; 是一个语句

在使用 var 声明变量, 同时完成赋值的时候. 实际上, 预解析将其做了一定处理:

-> 凡是读取到 var 的时候, 就检查 var 紧跟的名字是否已经标记了

-> 如果没有标记, 就表明这个名字是一个标识符, 需要被标记

-> 如果已经被标记了, 那么 这个 var 被忽略

结论:

var a;

var a = 10;

等价

var a;

a = 10;

var a = 123;

var a = 456;

var a = 789;

等价于

var a = 123;

a = 456;

a = 789;

如果在代码中有多个 var 后面紧跟的名字是一样的. 那么只有第一个 var 起作用.

后面的所有 var 都会被自动的忽略.

-> 变量名提升

代码再预解析的时候会去关注var声明的变量，函数的声明，标记这个变量或者函数

-> 函数的声明

-> 函数的各种定义形式

-> ①函数声明式:

function func () {

console.log( '使用声明式定义' );

}

-> ②函数表达式式( 匿名函数, 字面量函数, lambda 函数 ):

var func = function () {

console.log( '使用表达式式定义' );

};

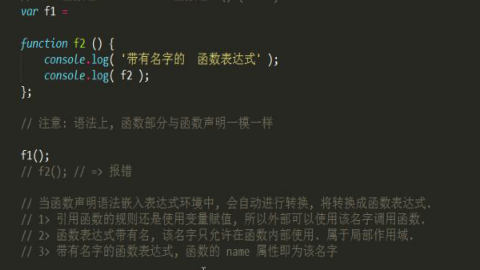
func();

使用这个方式定义函数，实际上是利用函数是js中的一个数据类型的特点，利用赋值，使用变量存储函数的引用，此时没有函数的声明，但是有变量的声明

--> ③带有函数名的函数表达式



f2外部无法使用只能在函数内部使用，如果要调用改函数要使用f1()调用，f2也是函数的name属性值



-> ...

-> 特点:

1> 函数的声明是独立于语句. 不需要加分号结束. 也不能嵌入到代码表达式中.

2> 表达式式, 本质上是使用函数表达式( 字面量 )给变量赋值. 因此它是语句.

-> 表达式:

-> 将运算符与操作数连接起来的式子.

-> 就是一个有结果的代码单元( 不包括语句 )

var a; // 声明, 不是语句, 也没有结果

123 // 字面量, 有值, 是表达式. 是常量表达式

a = 123 // 赋值, 有值, 就是被赋值的那个值. 是赋值表达式.

function () {}

函数名提升：代码预解析的时候会去找到声明的函数，提前把函数放在作用于里面

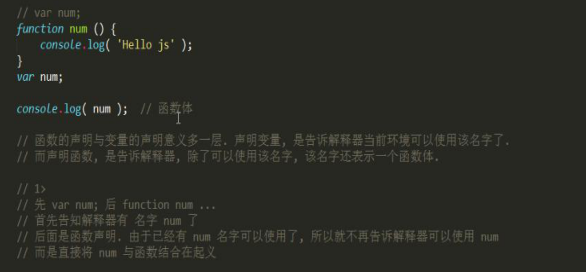
5. 如果将变量的声明与函数的声明放在一起有些需要注意的情况

1) 函数的声明实际上包含两部分

1> 告诉解释器 xxx 名字已经可以使用( 函数名, 标识符 )

2> 告诉解释, 这个名字代表着一个函数( 变量里存储着函数的引用 )

2) 当函数声明与变量声明冲突的时候. 只看谁先有数据.





6. 一个在新版本的浏览器中的特性

if ( true ) {

function foo() {

console.log( true );

}

} else {

function foo() {

console.log( false );

}

}

foo();

函数声明不要放在代码块里面 吧

7. 词法作用域

-> 作用域: 就是变量可以使用到不能使用的范围

-> 块级作用域:

-> 块: 代码块, 即 { }

-> 变量的使用从定义开始, 到其所在的块级作用域结束

-> js 是词法作用域

在js中，只有函数可以改变作用域的范围

-> 词法: 就是定义, 书写代码的规则.

-> 所以 所谓的 词法作用域, 就是 在书写代码的时候, 根据书写代码的结构

就可以确定数据的访问范围的作用域.

-> js 不受 块的影响, 即使在块中定义声明变量, 在块的外面依旧可以使用

console.log( num ); // => undefined

{

var num = 123;

}

console.log( num ); // => 123

-> 所谓的 js 的词法作用域, 就是

①在没有函数的情况下，所有的变量的访问规则依据预解析规则

②只有函数可以限定范围. 其他的不行.

③在函数内部是一个独立的完整的作用范围结构.

④在函数内部，允许再定义函数，同时两个层次的函数都是作用域的独立体

⑤在函数内优先访问内部声明的变量, 如果没有才会访问外部的.

⑥允许在函数内访问函数外的变量，前提是函数内没有改变量的声明

-> 结论:

词法作用域就是描述变量的访问范围:

1> 在代码中只有函数可以限定作用范围. 允许函数访问外部的变量. 反之不允许.

2>

3> 所有变量的访问规则, 按照预解析规则来访问

8. 案例

var num = 123;

function f1 () {

console.log( num );

}

function f2 () {

console.log( num );

var num = 456;

f1();

console.log( num );

}

f2();

1> 读取代码预解析. 得到 num, f1, f2

2> 逐步的执行代码

1) 赋值 num = 123; 注意 f1 和 f2 由于是函数, 所以也有数据.

2) 调用 f2.

进入到函数体内. 相当于做一次预解析. 得到 num. 注意, 此时有内外两个 num

执行每一句代码

-> 打印 num. 因为函数内部有声明 num. 所以此时访问的是函数内部的 num. 未赋值, 得到 undefined

-> 赋值 num = 456

-> 调用 f1(). 调用函数的规则也是一样. 首先看当前环境中是否还有函数的声明. 如果有直接使用. 如果

没有, 则在函数外面找, 看时候有函数. 此时在函数 f2 中没有 f1 的声明. 故访问的就是外面的 f1 函数

-> 跳入 f1 函数中. 又要解析一次. 没有得到任何声明.

-> 执行打印 num. 当前环境没有声明 num. 故在外面找. 外面的是 123. 所以打印 123.

函数调用结束, 回到 f2 中.

-> 继续执行 f2, 打印 num. 在 f2 的环境中找 num. 打印 456.

任务:

var num = 123;

function f1 () {

console.log( num );

}

function f2 () {

console.log( num ); // => 123 , 456, 456

num = 456;

f1();

console.log( num );

}

f2();

9. 案例

(function ( a ) {

console.log( a );

var a = 10;

console.log( a );

})( 100 );

拆解

( 函数 ) ( 100 )

第一个圆括号就是将函数变成表达式

后面一个圆括号就是调用该函数

var func = function ( a ) {

console.log( a );

var a = 10;

console.log( a );

}

func( 100 );

注意: 函数定义参数, 实际上就是在函数最开始的时候, 有一个变量的声明

function ( a ) { ... }

其含义就是, 在已进入函数体, 在所有操作开始之前( 预解析之前 )就有了该变量的声明.

由于已经有了 a 参数的声明. 所以在代码中 var a = 10 是重复声明. 其声明无效.

所以上面的代码, 等价于

var func = function ( a ) {

console.log( a ); // => 100

a = 10;

console.log( a ); // => 10

}

func( 100 );

// 变式

(function ( a ) {

console.log( a );

var a = 10;

console.log( a );

function a () {

console.log( a );

}

a();

})( 100 );

1> 直接调用

2> 进入到函数中, 已有声明 a 并且其值为 100

3> 在函数内部预解析. 得到 一个结论. 函数声明是两个步骤.

1) 让当前环境中, 有变量名 a 可以使用. 但是不需要. 因为已经有 a 的声明了

2) 让 a 指向函数. 相当于

var a;

function a () {}

...

4> 开始逐步执行每一句代码

1) 打印 a. 所以打印函数体

2) 赋值 a = 10

3) 打印 a, 打印出 10

4) 如果让 a 调用, 那么报错 error: a is not function

10. 作用域链规则

-> 什么是作用域链

链指的就是访问规则

function foo() {

console.log( num );

}

--------------------

function func () {

function foo() {

console.log( num );

}

foo();

}

--------------------

function F () {

function func () {

function foo() {

console.log( num );

}

foo();

}

func();

}

... ...

由于这种一环套一环的访问规则, 这样的作用域构成一个链式结构. 所以直接称其为作用域链.

-> 作用域链是用来做变量查找的. 因此变量可以存储什么东西. 链中就应该有什么东西.

换句话说就是, 链里面存储的是各种对象. 可以将其想象成对象的序列( 数组 )

11. 绘制作用域链的规则

1> 将所有的 script 标签作为一条链结构. 标记为 0 级别的链.

2> 将全局范围内, 所有的声明变量名和声明函数名按照代码的顺序标注在 0 级链中.

3> 由于每一个函数都可以构成一个新的作用域链. 所以每一个 0 级链上的函数都延展出 1 级链.

4> 分别在每一个函数中进行上述操作. 将函数中的每一个名字标注在 1 级链中.

5> 每一条 1 级链中如果有函数, 可以再次的延展出 2 级链. 以此类推.

12. 分析代码的执行

当作用域链绘制完成后. 代码的的分析也需要一步一步的完成.

1> 根据代码的执行顺序( 从上往下, 从左至右 )在图中标记每一步的变量数据的变化

2> 如果需要访问某个变量. 直接在当前 n 级链上查找变量. 查找无序.

3> 如果找到变量, 直接使用. 如果没有找到变量在 上一级, n - 1 级中查找.

4> 一直找下去, 知直到 0 级链. 如果 0 级链还没有就报错. xxx is not defined.

**第六天、闭包**

**一、相关概念**

1、闭包：闭包就是一个封闭的隔离的空间。js中：函数所构成的外部无法直接访问的区域

function foo () {}

闭包的本质：利用作用域访问规则的不可逆性，构成一个单向的空间。

为什么要学习闭包

因为有一些数据是比较敏感的，会有安全的需求，不能让用户得到，所以将它们放到闭包里面，让用户更加安全的访问。

**二、闭包的间接访问--访问闭包中的数据**

1、访问闭包中数据的方法：

在函数内部新定义一个函数，在这个新函数中return返回我们需要的原始数据，然后return返回这个函数。// 返回一个对象

2、案例：

function foo () {

var num = 123; // 原始数据

function func () {

return num; // 它就是原始数据 num

}

return func;

}

var f = foo(); // f中存储的是foo里面定义的函数的引用，即，可以用f来调用里面的函数

凡是调用一次f，就是调用一次foo中定义的函数，就会利用变量的词法作 用域规则返回foo中的num，即间接访问了闭包中的数据。

//foo只能调用一次，可以创建一个原始数据，但是声明的函数f可以重复调 用，每一次调用都可以获得闭包中的数据的值

**三、函数是对象--可以作为一般的数据来使用**

函数是基本的对象类型，可以作为变量赋值，可以作为参数使用，也可以作为返回值使用

函数返回函数：return 函数名；

赋值

function foo() {

......（函数体）

}

var func = foo; //函数体只有一个，是将函数的引用赋值给了func，因此，func和foo指向同一个函数，func和foo都可以调用该函数

函数作为参数进行传递，使用方法是调用----像正常的变量一样

function foo (fn) {

if (typeof fn === 'function'){

fn();

}

}

function func1() {}

var func2 = function () {};

foo(func1);

foo(func2); //函数名存储的函数引用，因此将函数作为参数传递时，直接传递函数名即可

函数作为返回值使用

function foo() {}

function func() {}

return func;

}

var f = foo();

f();

foo()();

function foo() {

return function() {};

}

**四、闭包应用**

1、 闭包代码的基本结构

function foo () {

var num = 123;

return function () {

return num;

}

}

2、如何返回函数中的多个数据

通过一个对象将返回的数据的多个方法包含起来，然后返回这个对象就可以了

请编写代码让外部可以访问到name，age，hobby

再编写代码让外部可以修改hobby

function func() {

var name="csw";

var hobby = "吃饭，睡觉";

return {

get\_name:function(){ return name; },

get\_hobby:function(){return hobby;},

set\_hobby:function(value){ hobby = value; }

};

}

var obj = func(); //obj保存的是返回的对象的引用，对象本身在func函数的空间里面

obj.set\_hobby("打豆豆");//通过set方法修改hobby。

var name = obj.get\_name();

3、闭包的核心内容

1> 带有私有数据的函数、对象

function foo () {

var num = 123;

return function () {

// 可以访问 num

}

}

var func = foo();

// 称 func 是一个 带有私有数据的 函数 // 称 func 带有缓存

2> 视情况提供访问数据的通道（返回的函数，返回的对象）

4、闭包的应用\_\_沙箱模式

沙箱就是一个隔离的执行环境

**沙箱模式：一个自调用的函数，我们写在这个函数里面的变量不会被外部的内容所影响到，这样代码即可以正常使用和运行，也会使我们的数据更加的安全。**

有时为了使得代码更加简洁, 会引入很多变量，定义变量越多, 出现冲突的可能性越大.

(function () {

// 沙箱模式

// 所有的代码写在这里

})();

5、闭包的应用\_\_模拟 window.onload 事件的追加与移除

o.addEvent( function ... )

o.removeEvent( fn )

模拟window.onload去制作一个对象，希望通过这个对象添加函数，删除函数，当页面加载完成以后，按照（私有数组存储函数）的顺序逐个的执行这些函数。

var itcastload = (function () {

var data = [];

window.onload = function () {

for ( var i = 0; i < data.length; i++ ) { // 依次执行 数组中的 方法

data[ i ]();

}

};

return {

addEvent: function ( fn ) {

data.push( fn );

},removeEvent:function( fn ){ //遍历 data，发现与fn相同的就删除

for(var i = data.length - 1;i>=0;i-- ){//删除以后arr.length就发生了变化，倒过来循环

if(fn === data[i]){

data.splice(i,1); //arr.splice(从第几个元素开始，删除的个数)

}

}

}

};

})();

itcastload.addEvent(function(){console.log()}; // 先执行

console.log(); // 后执行

function f () {console.log()}

itcastload.removeEvent(f){};

//itcastload.removeEvent(){function f () {console.log()}};

//console.log({} =={}); false!! 解释执行，解释成完全不同的对象

6、利用闭包模拟一个缓存结构

cache 对象, 可以使用 cache[ key ] = value 存储数据, cache[ key ] 获得数据，当 cahche 里面 的数据达到 1024 条, 将最早放进去的数据移除，cache = {} 可以存取数据, 但是不能限定数据的 长度，所以如果需要限定数据, 就是在加入数据的时候判断, 是否已超过尺寸，如果是, 则移除, 如果不是, 则什么也不做

将 cache 做成函数（函数本身也是对象）, 添加数据使用 cache( key, value )

function cache ( key, value ) {

（可以在这里加上限定长度的代码）

cache[ key ] = value;

}

cache( 'attr', function ( node, attr ) {

return node.getAttribute( 'name' ) == attr;

} );

cache( '\_\_name\_\_', '张三' );

//由于需要记录键的数量，并且需要记录添加数据的先后顺序，所有首先考虑有序的数组。

//因此需要让 cache 函数带有缓存功能（数组）

var cache = (function () {

var data = [] , max = 3;

function cache ( key, value ) {

if( data.length >= 3){//做判断, 如果超出范围, 则, 将最开始加入的 移除

var temp = data.shift();//将数组 第 0 项元素移除的 splice, shift

delete cache[temp];

}

data.push(key);

cache[ key ] = value;

}

return cache;

})();

cache(key, value);

//使用缓存

var v = cache (key);

if (!v) {

v = ...;

cache(key, v);

};

处理缓存的函数

function createCache () {

var data = [], max = 3;

function cache ( key, value ) {

if ( data.length >= 3 ) {

var temp = data.shift();

delete cache[ temp ];

}

data.push( key );

cache[ key ] = value;

}

return cache;

}

var kvCache = createCache();



**第七天**

**一、函数调用模式**

函数模式、方法模式、构造器模式、上下文模式

1、 函数模式

函数的定义方式

声明式 function func();

表达式式 var func = function(){};

Function new Function();

-> 单独独立调用的就是函数

函数名( 参数 )

-> this 表示全局对象

-> 任何自调用函数都是函数模式

语法：

function foo() {

。。。

console.log(this); //此时的this指的是window

}

foo();

2、方法调用 模式

方法本身就是函数, 但是方法不是单独独立的调用, 而是通过一个对象引导调用.

对象.方法( 参数 )

-> this 表示引导方法的对象

函数是独立的，方法需要对象引导，方法调用一定要有宿主对象（宿主对象可以是对象、函数、数组等），方法一定是某个对象的方法

语法：

function foo() {

console.log(this);

}

var o = { age: 19}; //这里o是宿主对象

o.foo = foo; //这里不能加()，如果加()就是调用foo函数的返回值赋值给o.foo

foo();

o.foo(); //此时this返回的是对象 o

function func () {

console.log(this);

}

func.foo = foo; //这里func是宿主对象

foo(); //函数调用

func.foo(); //方法调用，此时this返回的是函数func（函数整体）

3、构造器模式( constructor构造函数模式, 构造方法模式)

使用 new 关键字引导var p = new Person();

构造函数的执行过程，首先通过new创建一个对象，当创建出来对象以后，对象的原型结构就已经确定好了，但是对象里面没有任何成员，它实际上是一个空对象，当它调用构造函数的时候，它把对象的引用交给构造函数，构造函数里面的this就会指向这个函数，然后通过this去执行了this.name,this.age的赋值，实际上就是利用对象的动态特性给当前对象添加了成员

语法：

function Person () {

this.name = 'Jack';

this.age = 19;

}

var p = new Person();

-> 返回值

如果没有return 语句, 那么 构造函数 默认返回 this（即当前对象Person）

如果在构造函数中return 的是基本类型( return num, return 1223 ). 则忽略return,返回this

如果在构造函数中return 的是引用类型, 那么返回该引用类型数据, 而忽略 this

关于构造函数结合性的一个小结

1> 如果调用构造函数的时候构造函数没有参数, 可以省略 圆括号

var p = new Person;

2> 如果希望创建对象并直接调用其方法

( new Person () ).sayHello()

-> 可以省略调整结核性的圆括号

new Person().sayHello()

-> 如果想要省略构造函数的圆括号, 就必须添加结核性的圆括号

(new Person).sayHello()

4、上下文调用模式 === 环境调用模式

就是一种方法在不同环境下, 可以同时实现函数模式与方法模式，该方式调用不会污染宿主对象

apply 形式 函数名.apply( ... )

call 形式 函数名.call( ... )

上下文调用存在的目的就是为了实现借用方法

apply调用语法：

-->不带参数的上下文调用模式

function foo () {

console.log( this );

}

var o = { name: 'jim' };

foo(); //函数调用模式

o.func = foo;

o.func(); //方法调用模式

foo.apply( null ) 或 foo.apply() // 函数模式

foo.apply( o ) //方法模式, 注意需要提供一个宿主对象

-> 带有参数的函数如何实现上下文调用

function foo ( num1, num2 ) {

console.log( this );

return num1 + num2;

}

var res1 = foo( 123, 567 );// 函数调用模式

foo.apply(null, [123, 567); //apply函数调用

// 方法调用模式

var o = { name: 'jim' };

o.func = foo;

var res2 = o.func( 123, 567 );

foo.apply(o, [123, 567);//apply函数调用

函数模式：函数名.apply( null, [] );

方法模式：函数名.apply(对象名, []); //这里的对象就是宿主对象，函数的参数放在数组中

call 调用

在使用 apply 调用的时候, 函数参数, 必须以数组的形式存在. 但是有些时候数组封装比较复杂

call 调用与 apply 完全相同, 唯一不同是是参数不需要使用数组

foo( 123, 567 );

foo.apply( null, [ 123, 567 ] );

foo.call( null, 123, 567 );

5、借用构造方法实现继承

function Person ( name, age, gender ) {

this.name = name;

this.age = age;

this.gender = gender;

}

传统的继承

function Student () {}

Student.prototype = new Person();

借用构造方法的继承

function Student ( name, age, gender, course ) {

Person.call( this, name, age, gender );

this.course = course;

}

var p = new Student ( 'jim', 19, 'male', '前端' );

二、补充知识点

1、函数的 bind 方法（ES5）

bind 就是 让函数绑定对象的一种方法

函数本身就是可以调用，但是其如果想要作为方法调用，就必须传入宿主对象，并且使用call或apply形式，但是bind使得函数可以与某一个对象绑定起来，那么在调用函数的时候，就好像是该对象在调用方法

语法： 函数.bind (对象)

假如返回一个函数foo，那么调用返回的函数foo，就好像对象在调用该方法一样

var t = document.getElementsByTagName.bind(document);//让t包含函数体的同时也包含对象

arr = [];

arr.push.apply(arr, t('p'));

arr.push.apply(arr, t('p'));

for (var i = 0; i < arr.length; i++) {

arr[i].style.border = '1px solid red';

}

2、Object.prototype 的成员

constructor

hasOwnProperty 判断该属性是否为自己提供

propertyIsEnumerable 判断属性是否可以枚举 for in

isPrototypeOf 判断是否为原型对象

toString, toLocaleString, valueOf

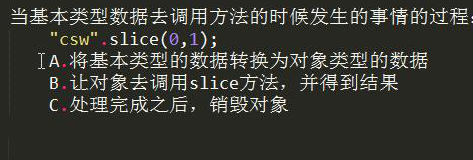
3、包装对象

在开发中常常会使用基本数据类型, 但是基本数据类型没有方法, 因此 js 引擎会在需要的时候，自动的将Number，String，Boolean三个基本类型转换成对象类型，可以使用charAt, substr, slice, ...等方法

执行过程：

"abc".charAt( 1 )

"abc" -> new String( "abc" )



.charAt( 1 ) 返回结果

s 就被销毁

当 基本类型.方法 的时候. 解释器首先将基本类型转换成对应的对象类型, 然后调用方法。 方法执行结束后, 这个对象就被立刻回收

在 apply 和 call 调用的时候, 也会有转换发生。上下文调用的第一个参数必须是对象. 如果传递的是数字就会自动转换成对应的包装类型

4、getter 与 setter 的语法糖

语法糖: 为了方便开发而给出的语法结构

var o = (function () {

var num = 123;

return {

get\_num: function () {

return num;

},

set\_num: function ( v ) {

num = v;

}

};

})();

o.get\_num(); //获得数据 => 希望o.num 形式，简单

o.set\_num( 456 ); //设置 => 希望o.num = 456 形式，简单

var o = (function () {

var num = 123;

return {

get num () { // get 名字 () { 逻辑体 }

return num;

}, 语法糖

set num ( v ) { // set 名字 ( v ) { 逻辑体 }

num = v;

}

};

})();

5、 ES5 中引入的部分数组方法

forEach遍历数组

语法：数组.forEach(fn) 例如 arr.forEach(function (v, i){};

map 映射，返回一个数组，数组的每一个元素就是map函数中的fn的返回值

语法：数组.map(fn(v, i)) v是数据，i是索引

filter 筛选

语法：数组.filter(function (v, i)){return true/false}

some

判断数组中至少有一个数据符合要求，就返回true，否则就返回false

every

必须满足所有的元素都符合要求才会返回true

indexOf

在数组中查找元素，如果含有该元素，返回元素的需求（索引），否则返回-1

lastIndexOf

从右往左找元素

例题：上下文对象的实际应用

需求：获得div与p标签，并添加边框border: 1px solid red

方法① var t = document.getElementsByTagName;

var p\_list = t.apply(document, ['div']);

var div\_list = t.apply(document, ['div']);

var arr = [];

[].push.apply(arr, p\_list);

[].push.apply(arr, div\_list);

for (var i = 0; i < arr.length; i++) {

arr[i].style.border = '1px solid red';

}

方法② var t = document.getElementsByTagName, arr = [];

arr.push.apply(arr, t.apply(document, ['p']));

arr.push.apply(arr, t.apply(document, ['div']));

arr.forEach(function (v){

v.style.border = '1px solid red';

})

方法③ var t = document.getElementsByTagName.bind(document);//让t包含函数体的同时也包含对象

arr = [];

arr.push.apply(arr, t('p'));

arr.push.apply(arr, t('p'));

for (var i = 0; i < arr.length; i++) {

arr[i].style.border = '1px solid red';

}

数组合并数组的方法

var arr1 = [1,2,3,4];

var arr2 = [5,6];

arr1.push(arr2); //这样是将arr2数组作为一个元素追加到arr1中，结果arr1[1,2,3,4,arr2]

[].push.apply(arr1, arr2);

面试题

var length = 10;

function fn() {

console.log( this.length );

}

var obj = {

length: 5,

method: function ( fn ) {

fn();

arguments[ 0 ]();//对象，接受传递的参数

}

};

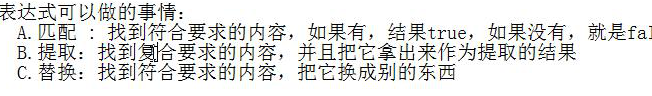
obj.method( fn, 1 );

**第八天**

1. 正则表达式

regular expression一个有规则的表达式

正则表达式就是 一个用于查找的 含有 匹配字符串 或 匹配元字符的 字符串



2. 正则表达式对象

js 正则表达式对象 就是 由 正则表达式创建的对象, 该对象可以进行 匹配, 提取 和 替换.

创建正则表达式对象

1) 构造函数

var regex = new RegExp( 正则表达式字符串[, 匹配模式] );

2) 字面量

var regex = /正则表达式/;

3. 使用正则表达式进行匹配

语法:

正则对象.test( 字符串 ) -> bool

如果参数字符串中含有 复合 正则匹配的 子字符串, 就返回 true, 否则返回 false

基本元字符

. 表示任一个非换行的字符

案例: foot \_ 是 food, foot 写成 foo.

() 表示分组和提高优先级

[] 表示一个字符, 出现在 [] 中的字符

用法: [abc] 出现 [] 中的任意一个字符

匹配 a, 或 b, 或 c

| 或

用法: 正则表达式|正则表达式

转义字符: \

表示点: \.

表示[]: \[ \]

表示(): \( \)

表示\ : \\

5. 限定元字符

\* 紧跟前面的一个字符或一组字符出现 0 次到多次

+ 紧跟在前面的字符出现 1 次到多次

? 紧跟在前面的字符出现 0 次或 1 次

{数字} 紧跟在前面的字符出现指定次数

a{3} aaa

{数字,} 紧跟在前面的字符至少出现指定次数

a{3,} aaa, aaaaaaaaa, aaaaaaaaaaaaaa, ...

{数字, 数字} 紧跟在前面的字符出现的次数范围

a{1, 3} a, aa, aaa

6. 首尾元字符

^ 表示必须以 xxx 开头

$ 表示 必须以 xxx 结尾

7. 简写元字符

\s 空白字符, 包括空格, tab, 回车换行等

\S 非空白字符

js 中常常使用 [\s\S] 表示任意字符

\w 表示字符, 包含字母, 数字, 下划线.

\W 非字符

\d 数字

\D 非数字

8. 提取

使用正则表达式可以进行匹配, 使用 exec 可以将匹配到的数据提取出来

语法:

正则表达式对象.exec( 字符串 ) -> 封装成数组

11. 循环提取

全局模式：匹配正则表达式的一种方式，将一个字符串中所有符合的要求的字符串提取出来，

1> 正则表达式需要使用全局模式

var r = new RegExp( '正则', 'g' );

var r = /正则/g;

2> 调用 exec 首先获得第一个匹配项，一直调用下去, 就可以得到所有的匹配项，直到最后全部匹配完, 如果还用该方法, 则返回 null

案例： 在 'abc123def456ghij789klum' 提取数字

var str= 'abc123def456ghij789klum';

var r = /\d+/g;

while ( res = r.exec( str ) ) {

// 操作 res, 处理捕获的结果

}

12. 将匹配到的结果进行解析

假如有一个字符串: itcast@itcast.cn

匹配提取邮箱, 同时将邮箱地址部分的名字和主机名都提取出来

13. 匹配但是不去捕获的元字符

(?:其他正则表达式内容)

14. 要截取一个字符串中的一个 html 标签

var str = '123<div>456</div>78<span>9</span>0';

1> 要截取 html 标签, 标签使用 <>. 因此正则表达式写成 <\w+>

2> 与之匹配的标签名是什么?

反向引用：如果在一个正则表达式中使用组匹配到某一个数据, 允许在该正则表达式中使用

'\数字' 的方式引用该组

<(\w+)>.\*<\/\1>

15. 贪婪模式

贪婪模式：凡是在正则表达式中, 涉及到次数限定的, 一般默认都是尽可能的多匹配.

取消贪婪模式： 在次数限定符（+，\*，？，{}）后面加上 ?

注意: 贪婪模式性能会略高于非贪婪模式, 所以开发的时候. 一般不考虑贪婪的问题

只有代码匹配结果出现问题了, 一般一次多匹配了, 才会取消贪婪

多个贪婪在一起的时候的强度

\d+\d+\d+

1234567

如果取消贪婪模式

\d+?\d+\d+

16. 否定元字符

语法:

[^字符]

不为这些字符的字符

[^abc] 不是 a 也不是 b 也不是 c 的字符

17. 字符串的替换

语法:

字符串.replace( 查找字符串, 替换字符串 ) -> 字符串

1> 字符串替换方法

字符串1.replace( 字符串2, 字符串3 )

在 字符串1 中找到 字符串2, 将其替换成 字符串3, 返回替换后的字符串

特点: 只替换第一个找到的字符串

2> 正则表达式替换

字符串.replace( 正则表达式, 字符串 ) -> 字符串

1. 简单替换

'aaaaa-------bbbbbb------ccccc'.replace( /-+/, '-' ) -->'aaaaa-bbbbbb------ccccc'

'aaaaa-------bbbbbb------ccccc'.replace( /-+/g, '-' ) -->'aaaaa-bbbbbb-ccccc'

2. 分组替换

可以在替换字符串中, 使用 $数字 来引用替换的数据

'abc123'.replace( /(\d+)/, 'd$1' ) -> 'abcd123'

'1991-1-1'

中国: 1991年1月1日

欧洲: 1/1/1991

... : 1991 1 1

'1991-1-19'.replace( /(\d+)-(\d+)-(\d+)/, '$1年$2月$3日' )

3> 函数参数用法(了解)

语法:

字符串.replace( 正则表达式, fn )

'我的邮箱是:abc@itcast.cn, 你的邮箱是: defghijk@itcast.cn'

.replace( /\w+@\w+(\.\w+)+/g, '\*' );

'我的邮箱是:abc@itcast.cn, 你的邮箱是: def@itcast.cn'

.replace( /(\w+)@(\w+(\.\w+)+)/g, function ( s, g1, g2, g3 ) {

return '\*';

} );

// 要求 只显示 第一个字符, 其余的都是用 \* 表示

'我的邮箱是:abc@itcast.cn, 你的邮箱是: defghijklim@itcast.cn'

.replace( /(\w+)@(\w+(\.\w+)+)/g, function ( s, g1, g2, g3 ) {

// 将用户名 变成 a\*\*\*\*\* 的形式

var first = g1.charAt( 0 );

var start = [];

for ( var i = 0; i < g1.length - 1; i++ ) {

start.push( '\*' );

}

return first + start.join('') + '@' + g2;

} );

7. 案例

1> 写一个正则表达式匹配 身份证号码

身份证是 18 位数字

省 市 区 出生年月 随机编码X

1) 首先是要做匹配, 就一定要使用 ^ $

2) 是 18 位数字( 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ). 可以使用 [0123456789]

还可以使用 0|1|2|3|4|5|6|7|8|9

3) 要求是 18 位. 限定 18 位, 即 {18}

组合: ^[0123456789]{18}$

如果最后一位是x

前面 17 位数字: ^[0123456789]{17}$

后面要么是数字: [0123456789]

要么是 x: x

写法: [0123456789x]

[0123456789]|x

0|1|2|3|4|5|6|7|8|9|x

^[0123456789]{17}[0123456789x]$

2> 匹配邮箱

12345678901@qq.com

abcdefg@126.com

abcdefg@163.com

abc@yahoo.com.cn

.cc

.org

.edu

.中文

...

名字 @ 主机名

1) 是要验证邮箱, 那么就需要使用 ^ $

2) 名字:

数字与字母

[0123456789]

[abcdefghi...]

[ABCDEFG...]

[] 里面的字符如果是编码序号连续的可以使用连字符连接

数字: [0-9]

[9-0] 错误的, 编码逆序

字母: [a-z]

[A-Z]

整合: [0-9a-zA-Z]

名字的变式方法: [0-9a-zA-Z]+

3) 主机名

主机名也是一串字符串或数字

但是它多了一个 .com .cn

3.1) 只有名字 [0-9a-zA-Z]+

3.2) 只含有一个 .什么

开始 \.

中间 [0-9a-zA-Z]+

只含有一个名字: \.[0-9a-zA-Z]+

3.3) 含有多个名字

.com.con.cc.c1.c2.c3

即 .什么 出现一次到多次

(\.[0-9a-zA-Z]+)+

最后主机名可以写成

[0-9a-zA-Z]+(\.[0-9a-zA-Z]+)+

最后整合一下

^[0-9a-zA-Z]+@[0-9a-zA-Z]+(\.[0-9a-zA-Z]+)+$

8. 匹配一个数字

1> 匹配一个数字

[0-9]+

1) 由于是匹配, 包含 ^ $

2) 首先第一个字符不允许是 0, 所以第一个可以写成 [1-9]

3) 后面的数字就是 [0-9]

4) 要求后面的数字出现 0 次到多次, 以匹配任意的 非 0 数字: [1-9][0-9]\*

5) 由于还需要考虑 0, 因此写成 [1-9][0-9]\*|0

6) 考虑 | 优先级最低: ^([1-9][0-9]\*|0)$

7) 考虑正负

^(-?[1-9][0-9]\*|0)$

2> 匹配一个指定范围的数字

匹配 0 ~ 255 为例

[0-255] 匹配的是0, 1, 2, 5

如果要匹配指定范围的数字, 那么需要将字符串结构进行分类

1) 0 要匹配的, 所以在正则表达式中有 0 这一项

2) 任意的 2 位数, 即 [1-9][0-9]

3) 任意的 1 位数, 即 [0-9], 可以将 第 1) 结论合并

4) 考虑 3 位数的时候, 只允许出现 1xx 的任意数, 而 2xx 的有限制

因此在分组, 考虑 1xx 的任意数, 可以写成: 1[0-9][0-9]

5) 考虑 2xx 的数字, 在 200 到 250 之间允许任意取. 所以

写成: 2[0-4][0-9]

6) 考虑 250 到 255, 写成 25[0-5]

综合一起:

^([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])$

匹配赢 IPv4

127.0.0.1

192.168.1.1

^(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]))$

3> 匹配小数

要求, 小数不允许以 0 结尾

3.1415

[0-9]\*[1-9]

^((-?[1-9][0-9]\*|0)(\.[0-9]\*[1-9])?)$

jq 中有一属性过滤选择器

[value=name] [value] [value !=name] ...

了解

\[[a-zA-Z]((!|\||~|^|$|)=...|)\]

案例:

var str = '我有一个邮箱, 是 itcast@itcast.cn, 还有 abc@126.com, 和 1234567@qq.com'

要求将里面的邮箱全部提取出来, 然后将名字还有主机名也取出来

// 循环提取的正则表达式

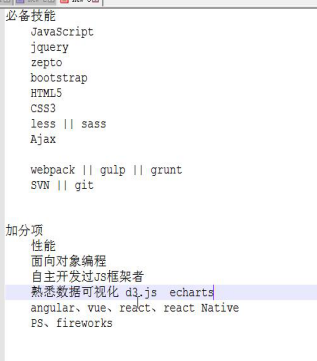
var r = /[a-zA-Z\d]+@[a-zA-Z\d]+(\.[a-zA-Z\d]+)+/g;

// 如果需要将匹配到的结果再次分解, 可以使用分组的形式, 在正则中分组即可

var r = /([a-zA-Z\d]+)@([a-zA-Z\d]+(\.[a-zA-Z\d]+)+)/g;

// 注意, 正则表达式中, 分组是有编号的. 从左往右数 '('. 从 1 开始依次编号

// 匹配提取的结果中, 对应的编号就可以获得分解的数据



**## AJAX第一天**

**一、服务器与客户端**

1、服务器和客户端说白了，都是电脑

服务器：能够提供某种服务的机器（计算机），主要给上网用户提供后台服务

客户端：具有向服务器索取服务能力的终端，主要用于普通上网用户

2、服务器

定义：使计算机具备提供某种服务能力的应用软件，称为服务器软件，通过安装相应的服务软件，然后进 行配置后就可以使计算具备了提供某种服务的能力。

**二、网络相关概念**

1. IP地址

IP就是给每个连接在互联网上的主机分配的一个32位地址。

查看本机的IP： ipconfig 或者ipconfig -all

查看域名的IP： ping

2. 域名

由于IP地址是基于数字，不方便记忆，于是便用域名来代替IP地址，域名就是方便IP的的别名，替代品

一个域名可以对应多个IP（负载均衡）

一个IP可以对应多个域名（虚拟主机）

3. 端口

端口是计算机与外界通讯交流的出口，每个端口对应不同的服务。说白了就是服务所对应的出口。

**三、静态网站**

1. 全部由HTML（标准通用标记语言的子集）代码格式页面组成的网站，所有的内容包含在网页文件中。

一般文件名均以htm、html、shtml等为后缀。

2. 优点：

a、静态网页的内容相对稳定，因此容易被搜索引擎检索

3. 缺点：

a、没有数据库的支持，在网站制作和维护方面工作量较大

b、静态网页的交互性较差，在功能方面有较大的限制 。

**四、动态网站**

1. 动态网站并不是指具有动画功能的网站，而是指网站内容可根据不同情况动态变更的网站，

一般情况下动态网站通过数据库进行架构。

一般动态网站体现在网页一般是以asp，jsp，php，aspx等结尾。

2. 优点：

a、动态网页以数据库技术为基础，可以大大降低网站维护的工作量。

b、交互能力强，网页会根据用户的要求和选择而动态改变和响应。

c、维护方便，即无需手动操作，便会自动生成新的页面

3. 缺点

a、动态网页中的“？”对搜索引擎检索存在一定的问题，

因此采用动态网页的网站在进行搜索引擎推广时需要做一定的技术处理才能适应搜索引擎的要求

**五、php基础语法**

基础

1. 所有的PHP相关的代码都要写在<?php … ?>里面

echo 的作用就是向页面当中输入字符串，换行 '<br>'

2. PHP 文件的默认文件扩展名是 ".php"。

3. PHP脚本在服务器上执行。AJAX也是

所有咱们现在写代码都要放在Apache服务器下打开

4. PHP 语句必须以分号结尾（;）

变量声明与字符串拼接

1. PHP的变量声明以及变量名的规则

PHP中定义变量使用的 $ 操作符；

变量命名的规则：字母数字下划线，不能以下划线开头，对大小写敏感

2. PHP中的字符拼接使用“ . ”，js中用“+”；

3. PHP中的单引号把包含在其中的变量当做普通的字符串来处理，不会给它赋值，直接输出

PHP中的双印号把包含在其中的变量当做变量解析成变量值，$变量 后面不能直接跟其他字符串

4、JSON字符串转化成对象：

var json = '{"name":"lily"}'; json中的数据必须用双引号

var object = JSON.parse(json数据);

数组

数组的两种形式：

$arr = array(1,2,3);默认key是从0 开始的整数

$arr = array("a"=>"1", "b"=>"2","c"=>"3");

echo $arr["a"];//1

1. echo --- 输出简单数据类型，如字符串、数值

2. print\_r($数组名) --- 输出复杂数据类型，如数组

3. var\_dump($数组名) --- 输出详细信息，如对象、数组

$arr = array(1,2,3);

echo $arr[0]; //1

print\_r($arr); //Array([0]=>1 [1]=>2 [2]=>3)

array (size=5) //0 => int 2

1 => int 3

2 => int 5

$arr1 = array('name' => 'lily, "age" =>'18');

echo $arr1['name']; //lily

print\_r($arr1); //Array([user]=>lily [age]=>18)

var\_dump($arr1); // array(size=2)

'user' => string 'lily' (length=4)

'age' => string '18' (length=2)s

二维数组

$arr = array();

$arr[0] = array(1,2,3);

$arr[1] = array(4,5,6);

print\_r($arr); //Array ( [0] => Array ( [0] => 1 [1] => 2 [2] => 3 ) [1] =>

Array ([apple]=>Array ([color] => red [shape] => round) )

数据类型与数组遍历

1. php的数据类型与JavaScript的数据类型是类似的，都是弱类型语言

字符串、整型、浮点型、布尔型、数组、对象、NULL

2. gettype() 内置函数，用来判断变量的类型

echo gettype($arr);

3. count()是内置函数，用来计算数组的长度

$arr = array(123,456,789);

for($i=0; $i<count($arr);$i++) {

echo $arr[$i].'<br>';

}

foreach($arr as $value) {

echo $value.'-----<br>';

}

foreach($arr as $key => $value){

echo $key.'===='.$value.'<br>';

}

**## AJAX第二天**

**一、php基础语法 ---- 函数**

1. 自定义函数，语法类似与JavaScript

2. 系统函数直接调用，不需要声明

3. json\_encode() --- 将数组和对象转换为字符串的方法;

php基础语法 ---- get请求参数获取

get[],post[]应用于表单处理

$\_GET是PHP内置好的专门用来接数据用的一个全局数组，它包含了表单提交上来的数据

1. http协议的常用请求方式：（增删改查）

a、get 用来从服务器获取数据（参数一般作为查询条件）

b、post 用来添加数据（表单）

c、put 用来修改数据

d、delete 用来删除数据

2. $\_GET['abc']得到url地址中传递的参数的值

3. 注意：绝不能使用 GET 来发送密码或其他敏感信息

**三、php基础语法 ---- post请求参数获取**

1. form 默认请求方式就是get请求，get请求会把表单数据作为url的参数进行提交

form 的method设置成post

2. 设置服务响应的文件类型：（如果请求的数据乱码，要在php文件中设置）

1. header("Content-Type: text/plain; charset=utf-8");

2. header("Content-Type: text/html; charset=utf-8");

3. $\_POST也是PHP内置好的专门用来接数据用的一个全局数组

GET和POST请求方式的差异

GET没有请求主体，使用xhr.send(null)

GET可以通过在请求URL上添加请求参数

POST有请求主体，可以通过xhr.send('name=itcast&age=10')

POST需要设置请求头

GET效率更好（应用多）

GET大小限制约4K，POST则没有限制

get 方式提交不太安全，post方式相对来说比较安全

**四、php基础语法----后台接口**

1. 将数组和对象转换为字符串的方法 --- json\_encode();

2. 将字符串转换为对象的方式 --- json\_decode();

3. 如何更好的理解接口这个概念

接口说白了就是后台返回特定格式数据，而不是一个完整的页面

就是从后台到前台返回一些数据

**五、原生Ajax实现页面局部更新**

什么是Ajax-----Asynchronous Javascript And XML(异步JavaScript和XML)，是一种创建交互式网页应用的网页开发技术，是多种技术的集合：JavaScript(XMLHTTPRequest) html css + 服务端

在不刷新页面的情况下，浏览器悄悄地、异步地向服务器发出HTTP请求。

服务器收到请求后，传回新的格式化数据回来（通常是JSON）。

浏览器解析JSON，通过DOM将新数据呈递显示，页面仅局部刷新

通过后台与服务器进行少量数据交换，Ajax可以使网页实现局部更新，这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。传统的网页如果需要更新内容，必须重载整个网页页面

**六、原生Ajax详解-xhr对象创建**

1. 标准下 --- xhr = new XMLHttpRequest();

2. IE6 --- xhr = new ActiveXObject('Microsoft.XMLHTTP');

兼容if(window.XMLHttpRequest){

xhr = new XMLHttpRequest();

else{

xhr = new ActiveXObject('Microsoft.XMLHTTP');

}

**七、原生Ajax详解-请求参数分析**

1. open()方法

xhr.open("get","test.txt",true);

调用open方法并不会真正发送请求，而只是启动一个请求以备发送。

它接受三个参数：要发送的请求的类型、请求的URL、表示是否异步的布尔值。

注意：如果是get请求，那么请求参数必须在url中传递

encodeURI()用来对中文参数进行编码，防止乱码。

post请求参数通过send传递，不需要通过encodeURI()转码，但是必须需要设置请求头参数

2. send()方法

如果要发送请求，用send()方法。

要发送特定的请求，需要调用send()方法。

它接受一个参数，即要作为请求主体发送的数据。

如果不需要通过请求主体发送数据，则必须传入null，不能留空。

get请求是只有头部，没有主体的；而post请求有请求主体。

3. xhr对象有一个重要的属性，就是readyState属性，表示“就绪状态”；

就是 --- xhr.readyState

readyState取值只有5种值：0、1、2、3、4

0 (uninitialized) 开始初始化

1 (loading) XMLHttpRequest已经发送了请求

2 (loaded) XMLHttpRequest浏览器已经收到了服务器响应的数据

3 (interactive) 正在解析数据

4 (complete) 数据解析完成，可以使用了



4. 只要readyState属性值发生了变化，就会触发一个事件onreadystatechange事件。

此时，可以用

xhr.onreadystatechange = function(){}

来捕获readyState改变之后做的事情。

5. Ajax一旦用send方法发出HTTP请求之后，

当readyState为4的时候，就会有一个属性产生 --- xhr.status

表示的是请求的文件的状态码

1\*\* ---- 消息

2\*\* ---- 代码请求成功 //200表示相应成功

3\*\* ---- 重定向

4\*\* ---- 请求错误 //404没有找到请求的资源

5\*\* ---- 服务器错误 //500服务器端错误

6. responseText

获得字符串形式的响应数据。

如果来自服务器的响应并非 XML，请使用 responseText 属性。

7. responseXML

获得 XML 形式的响应数据。

8. XMLHttpRequest 对象用于和服务器交换数据。

9. setRequestHeader --- 向请求添加 HTTP 头

xhr.setRequestHeader("Content-type","application/x-www-form-urlencoded");

**八、原生Ajax详解-响应状态分析**

原生Ajax详解：

1、创建XMLHttpRequest对象

var xhr = new XMLHttpRequest();

准备发送

encodeURI()用来对中文参数进行编码，防止乱码

var param = 'username' + uname + '&password' + pw;

xhr.open('get','./01check.php?'+encodeURI(param),true);

参数1、请求方式（get获取数据，post提交数据）

参数2、请求地址（如果是get请求那么请求参数必须在url中传递）

参数3、同步或者异步标志位，默认是true表示异步，false表示同步

(post请求)xhr.open('post','./01check.php',true); //post请求，参数通过send传递，不需要通过encodeURI转码

执行发送动作

xhr.send(null); //get请求这里需要添加null参数

(post请求)xhr.setRequestHeader('Content-Type','application/x-www-form-urlencoded);

(post请求)xhr.send(param); //post请求参数在这里传递，并且不需要转码

4、指定回调函数

xhr.onreadystatechange = function(){ //该函数调用的条件就是readyState状态从2-4

if(xhr.readyState == 4){ //表示服务器返回的数据已经可以使用了，但是这个数据不一定是正确的

if(xhr.status == 200){//表示服务器返回的数据是正常的，不是200表示数据是错误的

var data = xhr.responseText;

var info = document.getElementById('info');

if(data == '1'){

info.innerHTML = '登录成功';

}else if(data == '2'){

info.innerHTML = '用户名或者密码错误';

}

}

}

}

**### 第三天**

**一、json数据格式**

1、什么是JSON

- JavaScript 对象表示法

- 是存储和交换文本信息的语法

- 轻量级的文本数据交换格式

2、JSON数据和普通的JS对象的区别

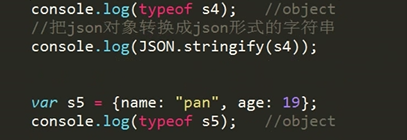
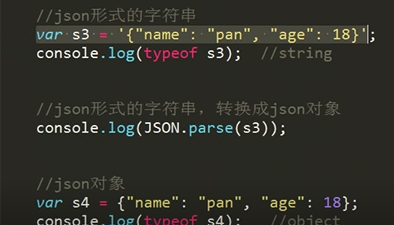
- json 对象没有变量

- json 形式的数据结尾没有分号

- json 数据中的键必须用双引号包括

-3、JSON和XML对比

- JSON 比 XML 更小、更快，更易解析



js对象，键没有引号

**二、json数据解析**

1、把JSON文本转换为JavaScript对象

JSON 最常见的用法之一，是从 web 服务器上读取 JSON 数据（作为文件或作为 HttpRequest），将 JSON 数据转换为 JavaScript 对象，然后在网页中使用该数据。

2、为什么要转换

在数据传输过程中，json是以文本，即字符串的形式传递的，而JS操作的是JSON对象，所以，JSON对 象和JSON字符串之间的相互转换是关键。

3、转换的方法

**JSON.parse() 把json形式的字符串转成对象**

JSON.stringify() 把对象转成字符串

eval() 把字符串解析成JS代码并执（不安全）

4、php开发json形式数据接口

**json\_encode() --- 把数组转换成JSON形式的字符串**

**三、同步与异步理解**

同步：等待一个请求完成，后续进程被阻塞，只有等请求返回后，才能执行后续的语句

异步：不等待请求完成，立即执行后续语句，请求返回后，再执行回调函数里面的语句

多线程：就是指计算机CPU有一定的物理结构，能够在物理上，同时计算两个东西。一边算加法，一遍算乘法。 程序是并发的。

单线程：就是计算机每时每刻，只有一个计算任务在执行。程序不是并发的。

同步与异步底层原理分析

js事件处理机制：

浏览器单线程，浏览器事件队列

事件队列中的任务执行的条件

定时函数（延时时间已经达到）

事件函数（特定事件被触发）

ajax的回调函数（服务器有数据响应）

拓展

单线程也能异步。就是你的程序只用一个线程，也就是说计算机同一时刻只能计算一个任务。那为什么还 能异步呢？？原因是，异步的时候，CPU把请求已经交给了磁盘这个物理硬件，而CPU还在以单线程运行。 比 如世界著名 Node.js语言，就是非常著名的单线程、异步的语言。

多线程也能同步。多线程，就是CPU能够同时计算多个命令，但是程序就是选择傻等，你管得着么？比 如 PHP是世界上最最注明的多线程语言，但是里面提供的读取文件方法，全是同步的。

**四、Ajax初步封装**

抽取函数 --- 学会抽取函数

实际开发中，重复利用的代码块也要封装成一个函数

理解回调函数的调用方式 --- 写5遍，写熟

array\_key\_exists() -- 这里的array\_key\_exists用来判断数组中没有对应键

**五、jQuery框架-Ajax相关API基本使用**

常用的jQuery ajax参数：

$.ajax({

type : "",

url : "",

data : "",

dataType : "",

beforeSend: function(xhr){

$('').html('正在拼命加载...');

}

success : function(data){

}

});

**## AJAX第四天**

一、同源策略

**1、什么是跨域**

同源策略是浏览器的一种安全策略，所谓同源指的是请求URL地址中的协议、域名和端口号都相同，只要其中之一不相同就是跨域。

同源策略主要是为了保证浏览器的安全性

在同源策略下，浏览器不允许Ajax跨域获取服务器数据

说简单点：跨域是指从一个域名的网页去请求另一个域名的资源。只要协议、域名、端口有任何一个的不同，就被当作是跨域。

2、跨域解决方案

jsonp，99%的公司都在使用jsonp。

利用script标签的跨域能力，这就是jsonp的基础。

**二、JSONP的原理（1）**

静态script标签的src属性进行跨域请求

如果使用src这种方法进行跨域的话，一定要把这段获取数据的代码在上面先行加载，使用数据的方法放 在后加载

这种方法的缺点就是非常的不灵活，这种方法已经极少使用，了解即可

JSONP的原理--动态（2）

利用js构造一个script标签，把json的url赋给script的src属性，把script插入到dom里，让浏览器去获取

动态创建script标签，通过标签的src属性发送请求

动态创建script标签发出去的请求是异步请求

服务器响应的内容是【函数调用】foo(实参)

**四、JSONP的原理（3）**

1、jsonp的本质：动态创建script标签，然后通过它src属性发送跨域请求，然后服务器响应的数据格式为【函数调用(foo(实参))】

所以在发送请求之前必须先声明一个函数，并且函数的名字与参数中传递的名字要一致，这里声明的函数是由服务器响应的内容，（实际就是一段js代码-函数调用）来调用

2、注意：ajax和jsonp其实本质上是不同的东西。

ajax的核心是通过XMLHttpRequest获取非本页内容，

而jsonp的核心则是动态添加<script>标签来调用服务器提供的js脚本。

**3、贺师俊 --- JSONP 的工作原理是什么**

就是利用<script>标签没有跨域限制的“漏洞”（历史遗迹啊）来达到与第三方通讯的目的。

当需要通讯时，本站脚本创建一个<script>元素，地址指向第三方的API网址，

形如： <script src="http://www.example.net/api?param1=1&param2=2"></script>

并提供一个回调函数来接收数据（函数名可约定，或通过地址参数传递）。

第三方产生的响应为json数据的包装（故称之为jsonp，即json padding），形如：

callback({"name":"hax","gender":"Male"})

这样浏览器会调用callback函数，并传递解析后json对象作为参数。本站脚本可在callback函数里处理所传入的数据。

4、定义函数，一定要写在调用JSONP之前。也就是，调用任何外部js，都没有函数声明头的提升。

jsonp属性的作用就是自定义参数名字（callback=abc,这里的名字指的是等号前面的键，后端根据这个键获取方法名，jQuery的默认参数名称是callback） //等号前面的是后台获取数据abc用的,GET(callback)

jsonpCallback属性的作用就是自定义回调函数的名字，（callback=abc，这里的名字指的是等号后面的值）

//等号后面的是定义回调函数的名字

**## AJAX第五天**

模板引擎：

模板 + 数据-->静态页面片段-->插入到页面中

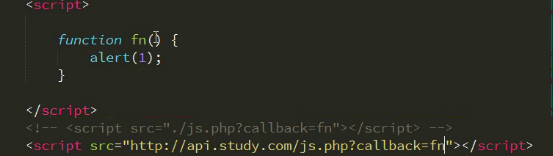
artTemplate 高效的原因：

1、预编译

artTemplate的编译赋值过程是在渲染之前完成的

2、更快的字符串相加方式

artTemplate根据JavaScript引擎特性采用了数组push拼接字符串和+=两种不同的字符串拼接方式，在IE6-8的浏览器下，数组push方法拼接字符串会比+=快，但是现代浏览器使用+=会比数组push方法快









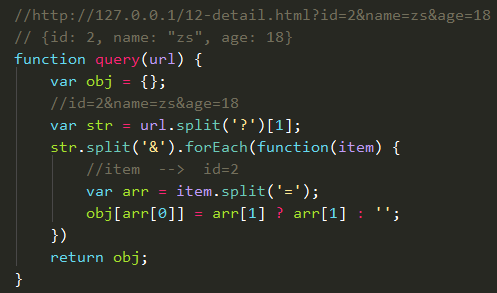
change([[data],fn])

当元素的值发生改变时，会发生 change 事件。

该事件仅适用于文本域（text field），以及 textarea 和 select 元素。当用于 select 元素时，change 事件会在选择某个选项时发生。当用于 text field 或 text area 时，该事件会在元素失去焦点时发生。

:get(index)所有大于0 的元素

**把？后面的参数取出来，方法**



Web前端与移动开发面试宝典

HTML+CSS部分

**1、怎么让一个不定宽高的DIV，垂直水平居中？**

1）使用CSS方法：占位置

父盒子设置：display：table-cell； text-align：center；vertical-align：middle；

Div设置： display：inline-block；vertical-align：middle；

2）使用CSS3的transform：不占位置

父盒子设置：position：relative

Div设置： transform： translate(-50%，-50%)；position： absolute；top： 50%；left： 50%；

**2、position几个属性的作用？**

position的常见四个属性值：relative， absolute， fixed， static。

一般都要配合"left"、 "top"、 "right" 以及 "bottom"属性使用。

1）Static：默认值，处于页面流给予的位置（static 元素会忽略任何 top、bottom、left 或 right 声明） 。

2）Relative：相对定位，可将其移至相对于其正常位置的地方，它偏移的top，right，bottom，left的值都以它原的位置为基准偏移。注意relative移动后的元素在原来的位置仍占据空间。

3）Absolute：绝对定位，可定位于相对于包含它的元素的指定坐标。意思就是如果它的父容器设置了position属性，并且position的属性值为absolute 或者relative，那么就会依据父容器进行偏移。如果其父容器没有设置position属性，那么偏移是以body为依据。注意设置absolute属性的元素在标准流中不占位置。

4）Fixed：位置被设置为 fixed 的元素，可定位于相对于浏览器窗口的指定坐标。不论窗口滚动与否，元

素都会留在那个位置。它始终是以body为依据的。 注意设置fixed属性的元素在标准流中不占位置。

**3、px，em，rem的区别？**

1）px像素（Pixel），绝对单位。像素px是相对于显示器屏幕分辨率而言的，是一个虚拟长度单位，是计 算机系统的数字化图像长度单位，如果px要换算成物理长度，需要指定精度DPI。

2）em是相对长度单位，相对于当前对象内文本的字体尺寸。如当前对行内文本的字体尺寸未被人为设置，

则相对于浏览器的默认字体尺寸。它会继承父级元素的字体大小，因此并不是一个固定的值。

3）rem是CSS3新增的一个相对长度单位，使用rem为元素设定字体大小时，相对的是HTML根元素。

4）区别：IE无法调整那些使用px作为单位的字体大小，em和rem可以缩放，rem相对的只是HTML根

元素。这个单位可谓集相对大小和绝对大小的优点于一身，通过它既可以做到只修改根元素就成比例 地调整所有字体大小，又可以避免字体大小逐层复合的连锁反应。目前，除了IE8及更早版本外，所 有浏览器均已支持rem。

**4、什么是BFC？**

1）定义：

BFC(Blockformattingcontext)直译为"块级格式化上下文"。它是一个独立的渲染区域，只有Block-levelbox参与， 它规定了内部的Block-levelBox如何布局，并且与这个区域外部毫不相干。

2）布局规则：

A.内部的Box会在垂直方向，一个接一个地放置。

B.Box垂直方向的距离由margin决定。属于同一个BFC的两个相邻Box的margin会发生重叠。

C.每个元素的marginbox的左边与包含块borderbox的左边相接触(对于从左往右的格式化，否则相反)。

即使存在浮动也是如此。

D.BFC的区域不会与floatbox重叠。

E.BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此。

F. 计算BFC的高度时，浮动元素也参与计算。

3）哪些元素会生成BFC：

A. 根元素

B.float属性不为none

C.position为absolute或fixed

D.display为inline-block， table-cell， table-caption， flex， inline-flex

F.overflow不为visible

**5、表格自动换行怎么实现？**

word-break：normal使用浏览器默认的换行规则；

break-all允许单词内换行；

keep-all只能在半角空格或连字符处换行

word-wrap：normal是用浏览器默认的换行规则；

break-word在长单词或URL地址内部进行换行。

**6、box-sizing、transition、translate分别是什么？**

Box-sizing： 用来指定盒模型的大小的计算方式。

boreder-box 从边框计算盒子大小

content-box 从内容固定盒子大小。

transition： 当前元素只要有“属性”发生变化时，可以平滑的进行过渡。

transtion-propety设置过渡属性；

transtion-duration设置过渡时间；

trantion-timing-function设置过渡速度；

trantion-delay设置过渡延时

translate：通过移动改变元素的位置；有x、y、z三个属性

**7、选择器优先级是怎样的？**

！important>行内样式>id选择器>类选择器>标签选择器>通配符>继承

权重算法：

（0，0，0，0）==》第一个0对应的是important的个数，第二个0对应的是id选择器的个数，第三个0 对应的类选择器的个数，第四个0对应的是标签选择器的个数，就是当前选择器的权重。

比较：

先从第一个0开始比较，如果第一个0大，那么说明这个选择器的权重高，如果第一个相同，比较第二个，依次类推

**8、CSS3选择器有哪些？**

属性选择器、伪类选择器、伪元素选择器。

**9、Iframe的作用？**

用法：

Iframe 是用来在网页中插入第三方页面，早期的页面使用iframe 主要是用于导航栏这种很多页面都相同的部分，这样可以在切换页面的时候避免重复下载。

优点：便于修改，模块分离，像一些信息管理系统会用到。

但现在基本上不推荐使用。除非特殊需要，一般不推荐使用。

缺点：

（1）iframe的创建比一般的DOM元素慢了1-2个数量级

（2）iframe标签会阻塞页面的加载，如果页面的onload事件不能及时触发，会让用户觉得网页加载 很慢，用户体验不好.在Safari和Chrome 中可以通过js动态设置iframe的src属性来避免阻塞.

（3）iframe对于SEO不友好，替代方案一般就是动态语言的Incude机制和ajax动态填充内容等.

**10、有一个导航栏在chrome里面样式完好？在IE里文字都聚到一起了，是哪个兼容性问题？**

用了display：flex属性，在ie10以下都是无效的。

**12、xhtml和html有什么区别？**

HTML是一种基本的WEB网页设计语言，XHTML是一个基于XML的置标语言

不同：XHTML 元素必须被正确地嵌套。必须被关闭。标签名必须用小写字母。必须拥有根元素。

**13、CSS引入的方式有哪些?link和@import的区别是?**

内联 内嵌 外链 导入

区别：同时加载 前者无兼容性，后者CSS2.1以下浏览器不支持，Link 支持使用javascript改变样式，后者不可

**14、标签上title与alt属性的区别是什么?**

Alt 当图片不显示时，用文字代表。 Title 为该属性提供信息。

**15、描述cssreset的作用和用途？**

Reset重置浏览器的css默认属性，浏览器的品种不同，样式不同，然后重置，让他们统一。

**16、解释csssprites，如何使用？**

Css 精灵图，把一堆小的图片整合到一张大的图片（png）上，减轻服务器对图片的请求数量。

**17、清除浮动的几种方式？**

1. 父级div定义 height

原理：父级div手动定义height，就解决了父级div无法自动获取到高度的问题。简单、代码少、容易掌握，但只适合高度固定的布局.

2. 结尾处加空div标签 clear：both

原理：在浮动元素的后面添加一个空div兄弟元素，利用css提高的clear：both清除浮动，让父级div能自动获取到高度，如果页面浮动布局多，就要增加很多空div，让人感觉很不好.

3. 父级div定义伪类：after 和 zoom /\*清除浮动代码\*/

.clearfix:after{

content：""；

display：block；

visibility：hidden；

height：0；

line-height：0；

clear：both；

}

.clearfix{zoom：1}

原理：IE8以上和非 IE 浏览器才支持：after，原理和方法2有点类似，zoom(IE 专有属性)可解决 ie6，ie7浮动问题， 推荐使用，建议定义公共类，以减少 CSS代码。

4. 父级div定义 overflow：hidden 超出盒子部分会被隐藏，不推荐使用.

5.双伪元素法：

.clearfix：before，.clearfix：after {

content： ""；

display： block；

clear： both；

}

.clearfix { zoom： 1；}

**18、z-index说说z-index的工作原理及适用范围 ?**

原理：

z-index这个属性控制着元素在z轴上的表现形式。 该属性仅适用于定位元素。即拥有relative，absolute，fixed属性的position元素。堆叠顺序（StackingLevel），是当前元素位于z轴上的值。数值越大表明元素的堆叠顺序越高，越靠近屏幕。未定义时，后来居上。如果未指定z-index的属性，元素的堆叠顺序基于它所在的文档树。默认情况下，文档中后面声明的元素具有更高的堆叠顺序。当父元素的堆叠顺序被设置的时候，这也意味着，它的子元素的堆叠顺序不能高于或低于这一顺序 (相对于父元素的堆叠上下文)。

适用范围：

网页两侧浮动窗口(播放器，置顶按钮，浮动广告，功能按钮等)

导航栏浮动置顶。

隐藏div实现弹窗功能(通过设置div的定位和z-index控制div的位置和出现隐藏)

**19、能否简述一下如何使一套设计方案，适应不同的分辨率，有哪些方法可以实现？**

流式布局：也就是百分比布局(viewport)， 使用非固定像素来定义网页内容，通过盒子的宽度设置成百分比来根据屏幕的宽度来进行伸缩，不受固定像素的限制，内容向两侧填充。

响应式开发： 就是一个网站能够兼容多个终端。CSS3中的Media Query（媒介查询)通过查询screen的宽度来指定某个宽度区间的网页布局。 由于响应式开发显得繁琐些，一般使用第三方响应式框架来完成，比如bootstrap来完成一部分工作。

**响应式开发和移动web开发的区别**

|  |  |  |
| --- | --- | --- |
| 开发方式 | 移动web开发+PC开发 | 响应式开发 |
| 应用场景 | 一般在已经有PC段的网站，开发移动站的时候，只需单独开发移动端 | 针对新建站的一些网站，现在要求适配移动端，所以就一套页面兼容各种终端，灵活 |
| 开发 | 针对性强，开发效率高 | 兼容各种终端，效率低 |
| 适配 | 只是陪移动设备，pad上体验相对较差 | 可以适配各种终端 |
| 效率 | 代码简洁，加载快 | 代码相对复杂，加载慢 |

rem

flexble

**20、你能描述一下渐进增强和优雅降级之间的不同吗？**

优雅降级graceful degradation：认为应该针对那些最高级、最完善的浏览器来设计网站，一开始就构建完整的功能，然后再针对低版本浏览器进行兼容

渐进增强progressive enhancement：针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验

区别：优雅降级是从复杂的现状开始，并试图减少用户体验的供给，而渐进增强则是从一个非常基础的能够起作用的版本开始，并不断扩充，以适应未来环境的需要。优雅降级意味着往回看，而渐进增强意味着朝前看，同时保证其根基处于安全地带。

**21、改变元素的外边距用什么属性？改变元素的内填充用什么属性？**

改变元素的外边距用margin，改变元素的内填充用padding。

**22、在新窗口打开链接的方法是？**

target：\_blank。

**23、合理的页面布局中常听过结构与表现分离，那么结构是什么？表现是什么？**

结构是html，表现是css。

**24、简述对Web语义化的理解？**

就是让浏览器更好的读懂你写的代码，在进行HTML结构、表现、行为设计时，尽量使用语义化的标签，使程序代码简洁明了，易于进行Web操作和网站SEO。

**25、每个HTML文件里开头都有个很重要的东西，Doctype，知道这是干什么的吗？**

DOCTYPE是一种标准通用标记语言的文档类型声明，它的目的是要告诉标准通用标记语言解析器，它应该使用什么样的文档类型定义来解析文档。只有确定了一个正确的文档类型，超文本标记语言或可扩展超文本标记语言中的标签和层叠样式表才能生效。

**26、CSS都有哪些选择器？CSS选择器的优先级是怎么样定义的？**

！important>行内样式>ID> 类 > 标签 | 伪类 | 属性选择 > 伪对象 > 继承 > 通配符。

**27、display： none；与visibility： hidden的区别是什么？**

答：display：none； 不占位置

visibility：hidden； 占位置

**28、请用CSS定义p标签，要求实现以下效果：字体颜色在IE6下为黑色(#000000)； IE7下为红色(#ff0000)；**

**而其他浏览器下为绿色(#00ff00)**

答：p{ color：green； \*color：blue； \_color：black； }

**二、JS部分**

**1、AJAX请求数据步骤是什么？传输的数据是用的暗文还是明文？ //明文**

**原生Ajax详解：**

1、创建XMLHttpRequest对象

var xhr ;

if(XMLHttpRequest){

xhr = new XMLHttpRequest()

}else{

xhr = new ActiveXObject("Microsoft.XMLHTTP");

}

2、设置请求地址，请求的方式，准备发送

var param = 'username' + uname + '&password' + pw;

xhr.open('get','./01check.php?'+encodeURI(param),true);

参数1、请求方式（get获取数据，post提交数据）

参数2、请求地址（如果是get请求那么请求参数必须在url中传递）

参数3、同步或者异步标志位，默认是true表示异步，false表示同步

(post请求)xhr.open('post','./01check.php',true); //post请求，参数通过send传递，不需要通过encodeURI转码

3、执行发送动作

xhr.send(null); //get请求这里需要添加null参数

(post请求)xhr.setRequestHeader('Content-Type','application/x-www-form-urlencoded);

(post请求)xhr.send(param); //post请求参数在这里传递，并且不需要转码

4、指定回调函数

xhr.onreadystatechange = function(){ //该函数调用的条件就是readyState状态从2-4

//XMLHttpRequest的状态

if(xhr.readyState == 4){ //表示服务器返回的数据已经可以使用了，但是这个数据不一定是正确的

if(xhr.status == 200){//表示服务器返回的数据是正常的，不是200表示数据是错误的

var data = xhr.responseText;

var info = document.getElementById('info');

if(data == '1'){

info.innerHTML = '登录成功';

}else if(data == '2'){

info.innerHTML = '用户名或者密码错误';

}

}

}

}

XMLHttpRequest对象会经历5种不同的状态。

//0：未初始化。对象已经创建，但还未初始化，即还没调用open方法；

//1：已打开。对象已经创建并初始化，但还未调用send方法；

//2：已发送。已经调用send 方法，但该对象正在等待状态码和头的返回；

//3：正在接收。已经接收了部分数据，但还不能使用该对象的属性和方法，因为状态和响应头不完整；

//4：已加载。所有数据接收完毕

**2、怎么实现跨域问题？**

对于端口和协议的不同，只能通过后台来解决。我们要解决的是域名不同的问题。

1. 下面是用php行的设置，“ \* ”号表示允许任何域向我们的服务端提交请求 header{"Access-Control-Allow-Origin： \*"}

2.静态script标签的src属性进行跨域请求

如果使用src这种方法进行跨域的话，一定要把这段获取数据的代码在上面先行加载，使用数据的方法放 在后加载

这种方法的缺点就是非常的不灵活，这种方法已经极少使用，了解即可

1. JSONP(JSONwithPadding 填充式JSON 或参数式JSON)

动态创建script标签，通过它的src属性发送跨域请求，服务器响应的数据格式为【函数调用(foo(实参))】

**JSONP 的工作原理是:利用<script>标签没有跨域限制的“漏洞”来达到与第三方通讯的目的。**

当需要通讯时，本站脚本创建一个<script>元素，地址指向第三方的API网址，

形如： <script src="http://www.example.net/api?param1=1&param2=2"></script>

并提供一个回调函数来接收数据（函数名可约定，或通过地址参数传递）。

第三方产生的响应为json数据的包装（故称之为jsonp，即json padding），形如：

callback({"name":"hax","gender":"Male"})

这样浏览器会调用callback函数，并传递解析后json对象作为参数。本站脚本可在callback函数里 处理所传入的数据。

优点： ①兼容性好，在更加古老的浏览器中都可以运行，不需要XMLHttpRequest或ActiveX的支持；

②能够直接访问响应文本，支持在浏览器与服务器之间双向通信

缺点： ①安全性不能保证

②只支持GET请求而不支持POST等其它类型的HTTP请求；

③只支持跨域HTTP请求这种情况，不能解决不同域的两个页面之间如何进行JavaScript调用的问题

1. **谈谈js作用域和闭包？**

作用域:是指对某一变量和方法具有访问权限的代码空间，也就是变量或函数起作用的区域。

包括全局作用域和局部作用域

全局作用域：任何地方都能访问到的对象拥有全局作用域

1. 最外层函数和变量
2. 未定义直接赋值的变量
3. window对象的属性

作用域链：在查找变量的时候会先从自身的作用域区查找，找不到再到上一级的作用域查找，如果还没找到就 到全局作用域区查找，这样就形成了一个作用域链。

闭包：如果某个函数被他的父函数之外的一个变量引用，就会形成闭包。 闭包的作用，就是保存自己私有的变量，通过提供的接口（方法）给外部使用，但外部不能直接访问该变量。

**4、什么是原型链？**

答：Javascript是面向对象的，每个实例对象都有一个\_\_proto\_属性，该属性指向它原型对象，这个实例对象的构造函数有一个原型属性prototype，与实例的\_\_proto\_\_属性指向同一个对象。当一个对象在查找一个属性的时， 自身没有就会根据\_\_proto\_\_ 向它的原型进行查找，如果都没有，则向它的原型的原型继续查找，直到查到 Object.prototype.\_proto\_为nul，这样也就形成了原型链。

原型链结构：实例对象 -> 对象的原型对象 -> object的原型对象 ->null 链式结构( 原型链 )

1. **实现继承的方法有什么？**

**实现继承，首先要有一个父类**

function Person ( name, age) {

this.name = name; //属性

this.age = age;

this.sayHello= function(){ //实例方法

console.log('你好！');

};

}

Person.prototype.eat= function(food){ //原型方法

console.log(this.name +'正在吃'+ food);

}

（1）原型链继承。

思路：借助原型可以基于已有的对象创建对象，将父类的实例作为子类的原型

function Student () {}

Student.prototype = new Person();

Student.prototype.name = '张三';

var stu2 = new Student();

console.log(stu2.name); //张三

console.log(stu2.eat('fish'); //张三在吃鱼

优点：

1. 非常纯粹的继承关系，实例是子类的实例，也是父类的实例
2. 父类新增原型方法/原型属性，子类都能访问到
3. 简单，易于实现

缺点：

* + - 1. 要想为子类新增属性和方法，必须要在new Animal()这样的语句之后执行，不能放到构造器中
      2. 无法实现多继承
      3. 来自原型对象的引用属性是所有实例共享的（详细请看附录代码： [示例1](http://www.cnblogs.com/humin/p/javascript:void(0);)）
      4. 创建子类实例时，无法向父类构造函数传参

（2）借用构造函数。使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类

function Student ( name, age ) {

Person.call( this, age);

this.name = name || '李四';

}

var stu3 = new Student ( 'jim', 19);

思路：在子类构造函数的内部调用超类型构造函数。可以通过使用apply（）和call()方法在新创建的对象上 执 行构造函数。

优点：

1. 解决了1中，子类实例共享父类引用属性的问题
2. 创建子类实例时，可以向父类传递参数
3. 可以实现多继承（call多个父类对象）

缺点：

1. 实例并不是父类的实例，只是子类的实例
2. 只能继承父类的实例属性和方法，不能继承原型属性/方法
3. 无法实现函数复用，每个子类都有父类实例函数的副本，影响性能
4. 实例继承。为父类实例添加新特性，作为子类实例返回

function Student (name){

var instance = new Person();

instance.name = name || '李四';

return instance;

}

var stu4 = new Student();

优点：不限制调用方式，不管是new 子类()还是子类(),返回的对象具有相同的效果

缺点：

a、实例是父类的实例，不是子类的实例

b、不支持多继承

1. 拷贝继承

function Student(name){

var per = new Person();

for (var p in per){

Student.prototype[p] = per[p];

}

Student.prototype.name = name || '李四';

};

var stu5 = new Student();

优点：支持多继承

缺点：

a、效率较低，内存占用高（因为要拷贝父类的属性）

b、无法获取父类不可枚举的方法（不可枚举方法，不能使用for in 访问到）

**（5）组合继承。**指的是将原型链和借用构造函数的技术组合到一起，通过调用父类构造，继承父类的属性并保留传参的优点，然后通过父类实例作为子类原型，实现函数复用。

function Student ( name, age ) {

Person.call( this,age);

this.name = name || '李四';

}

Student.prototype = new Person();

var stu3 = new Student ( 'jim', 19);

优点：

1. 弥补了方式2的缺陷，可以继承实例属性/方法，也可以继承原型属性/方法
2. 既是子类的实例，也是父类的实例
3. 不存在引用属性共享问题
4. 可传参
5. 函数可复用

缺点：调用了两次父类构造函数，生成了两份实例（子类实例将子类原型上的那份屏蔽了）

思路：使用原型链实现对原型属性属性和方法的继承，通过借用构造函数来实现实例属性的继承。 既通过在原型上定义方法实现了函数复用，又能保证每一个实例都有它自己的数组。 组合继承避免了原型链和借用构造函数的缺陷，融合了他们的优点，成为JavaScript中常用的继承模式。

1. 寄生式继承。

思路：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，最后再像真的是它做了 所有的工作一样返回对象。

function Student(name){

var clone = Person(name)；//通过调用函数创建一个新对象

clone.sayHi=function(){ //以某种方式来增强这个对象

alert(“hi”)；

}

return clone； //返回这个对象

}

var stu6 = Student();

缺点：使用寄生式继承来为对象添加函数，会由于不能做到函数复用二降低效率，这一点和构造函数模式类 似。

**（7）寄生组合式继承。**是JavaScript最常用的继承模式。

function Student ( name) {

Person.call( this);

this.name = name || '李四';

}

(function(){

// 创建一个没有实例方法的类

var Super = function(){};

Super.prototype = Person.prototype;

//将实例作为子类的原型

Student.prototype = new Super();

})();

var stu3 = new Student ( 'jim', 19);

思路：通过借用构造函数来继承属性，通过原型链的混入形式来继承方法。 本质上，就是使用寄生式继承来继承超类型的原型，然后再将结果指定给子类型的原型。

（8）混合式继承

把多个对象的成员混入到对象的原型对象中，然后构造函数创建的新对象就拥有了多个对象的成员

function extend( obj(原对象), obj1(新对象) ) {//将obj1的成员加入到obj中去

// 如何获得 obj1 中的所有成员?

for ( var k in obj1 ) {

obj[ k ] = obj1[ k ];

}

}

**6、什么是事件冒泡/捕获？**

事件冒泡： 事件到达事件目标之后不会结束，会逐层向上冒泡，直至document对象，也就是子元素事件的触发会影响父元素的事件；

开关事件冒泡：

A，开启事件冒泡：event.addEventListener(eventName，handler，false)；

B，关闭事件冒泡：假设传统方式事件的返回值为e，就可以通过e.stopPropagation()来关闭事件冒泡；

C ,return false; 同时具有阻止冒泡和阻止默认行为的功能

事件捕获：父元素的事件会影响子元素的事件；

开启事件捕获：event.addEventListener(eventName，handler，true)

**7、请说说事件委托机制？这样做有什么好处？**

事件委托，就是某个事件本来该自己干的，但是自己不干，交给别人来干。就叫事件委托。

打个比方：一个 button对象，本来自己需要监控自身的点击事件，但是自己不来监控这个点击事件，让自己的父节点来监控自己 的点击事件。

优点：

A，提高性能：列如，当有很多li同时需要注册事件的时候，如果使用传统方法来注册事件的话，需要给每 一个li注册事件。然而如果使用委托事件的话，就只需要将事件委托给该一个元素即可。这样就能提高性能。

B，新添加的元素还会有之前的事件；

**8、请列举字符串操作的方法？**

charCodeAt() 方法返回一个整数，代表指定位置字符的Unicode编码；

charAt() 方法返回指定索引位置处的字符。如果超出有效范围的索引值返回空字符串；

slice() 方法返回字符串的片段；

substring() 方法返回位于String对象中指定位置的子字符串。

substr() 方法返回一个从指定位置开始的指定长度的子字符串。

indexOf() 方法返回String对象内第一次出现子字符串位置。如果没有找到子字符串，则返回-1；

lastIndexOf() 方法返回String对象中字符串最后出现的位置。如果没有匹配到子字符串，则返回-1；

search() 方法返回与正则表达式查找内容匹配的第一个字符串的位置。

concat() 方法返回字符串值，该值包含了两个或多个提供的字符串的连接；

split() 将一个字符串分割为子字符串，然后将结果作为字符串数组返回；

**9、请说出==和===的区别？**

== 判断内容是否相等 不比较类型 console.log(1 == "1")； true

=== 判断内容相等 且类型也相等 console.log(1 ==="1")； false

**10、如何判断一个数组是数组？**

1.[] instanceof Array true/false

2. arr.constructor == Array

3. ES5：Array.isArray()

4. Object.prototype.toString.call([])==="[object Array]"

1. 目前完美方法

function isArray(value){

if (typeof Array.isArray === "function") {

return Array.isArray(value)；

}else{

return Object.prototype.toString.call(value) === "[object Array]"；

}

}

**11、怎么理解jQuery？**

JQuery就是JavaScript和查询（Query），是一个Javascript库。可以用来操作文档对象、 选择DOM元素、制作动画效果、事件处理、使用Ajax等。

优点：

1. 轻量级
2. 具有强大的选择器
3. 封装了大量的DOM操作
4. 事件处理机制相当可靠
5. 完善的Ajax
6. 不污染顶级变量
7. 出色的浏览器兼容性
8. 链式操作方式
9. 隐式迭代
10. 行为与结构分离
11. 支持丰富的插件
12. 完善的文档
13. 开源

**12、****Jquery.on这个方法怎么看？**

答： jQuery.on()方法可以给匹配元素（可以是多个）绑定一个或多个函数, off 可以解除绑定

**13、表单验证传输的什么数据？明文还是暗文==加密？如何加密？是每一次传输数据，都是加密之后才**

**传输吗？**

GET是从服务器上请求数据， POST是发送数据到服务器。事实上，GET方法是把数据参数队列（querystring） 加到一个URL上，值和表单是一一对应的。比如说，name=John。在队列里，值和表单用一个&符号分开，空格用+号替换，特殊的符号转换成十六进制的代码。因为这一队列在URL里边，这样队列的参数就能看得到，可以被记录下来，或更改。通常GET方法还限制字符的大小（大概是 256字节 ） 。事实上POST方法可以没有时间限制的传递数据到服务器，用户在浏览器端是看不到这一过程的，所以POST 方法比较适合用于发送一个保密的（比如信用卡号）或者比较大量的数据到服务器。

区别： Post是允许传输大量数据的方法，而Get方法会将所要传输的数据附在网址后面，然后一起送达服务器，因 此传送的数据量就会受到限制，但是执行效率却比Post方法好。

总结： 1、get方式的安全性较Post方式要差些，包含机密信息的话，建议用Post数据提交方式；

2、在做数据查询时，建议用Get方式；而在做数据添加、修改或删除时，建议用Post方式； 所以： 表达如果是向服务器传输数据(如帐号密码等)都是加密数据(post)，如果只是单单想要从服务器获得数据或者 传输的数据并不重要， 可以直接使用明文方式传输(get)

**14、面向对象和类的区别？**

答：简单的说类是对象的模版。 在 js 中没有类， 所以在js 中所谓的 类 就是构造函数， 对象就是由构造函数创建出来的实例对象。面向对象就是使用面向对象的方式处理问题， 面向对象是对面向过程进行封装。

面向对象有三大特性

1. 抽象性， 需要通过核心数据和特定环境才能描述对象的具体意义
2. 封装性， 封装就是将数据和功能组合到一起， 在js中对象就是键值对的集合， 对象将属性和方法封装起来， 方法将过程封装起来
3. 继承性， 将别人的属性和方法成为自己的， 传统继承基于模板(类)， js中继承基于 构造函数

对象：面向对象编程的程序实际就是多个对象的集合，我们可以把所有的事物都抽象成对象，在程序 设计中可以看作：对象=属性+方法。属性就是对象的数据，而方法就是对象的行为。

类：类是对象的模版，而对象是类的实例化。举个例子，汽车设计图可以看作是类，而具体的汽车就 是对象。再比如有一个类是表示人，然后可以通过人这个模版来实例化出张三、李四。 。 。

**15、在JS的计时器运行原理是怎样的，为什么可以触发计时效果? 计时器是多线程吗?**

1.javascript引擎只有一个线程，强迫异步事件排队等待被执行。

1. setTimeout和setInterval本质上不同的地方是他们如何执行异步代码的。

3.如果一个定时器正在执行的时候被阻塞了，那么它将会被推迟到下一个可能的执行点，这既是使得延迟时间有可能会超过声明定时器时设置的值。

4.Interval如果有足够的时间来执行（大于制定的延迟） ，那么它将会无延迟的一个紧接着一个执行。

原理： 计时器通过设定一定的时间段（毫秒）来异步的执行一段代码。因为 Javascript 是一个单线程语言，计时器 提供了一种绕过这种语言限制来执行代码的能力。

总结： 计时器是单线程的， 需要等待上一个执行完， 如果上一个没有执行完， 下一个需要延迟执行， 直到上一个执行完

**16、如何查找构造函数和原型中的属性？**

构造函数.prototype 查看构造函数的原型属性

实例对象.\_\_proto\_\_ 查看实例对象的构造函数的原型

实例对象.\_\_proto\_\_.constructor 查看实例对象的构造函数

**17、js中一共有几种数据类型？**

答：Undefined、Null、Boolean、Number和String。 还有一种复杂的数据类型Object，Object本质是一组无序的名值对组成的。

**18、call和apply的区别**

相同点：可以用来代替另一个对象调用一个方法，将一个函数的对象上下文从初始的上下文改变为由 thisObj 指定的新对象。

不同点：

apply：最多只能有两个参数——新this对象和一个数组 argArray。并且要参数要放到数组里 func.apply(func1，[var1，var2，var3])。

call：则是直接的参数列表，不用放到数组里func.call(func1，var1，var2，var3)

**19、说说你对this的理解？**

this是一个关键字，它代表函数运行时，自动生成的一个内部对象，只能在函数内部使用。

1. 作为纯粹的函数调用 this指向全局对象

2.作为对象的方法调用 this指向调用对象

3.作为构造函数被调用 this指向新的对象（new会改变this的指向）

4.apply调用 this指向apply方法的第一个参数

**20、谈谈你对递归的认识？**

递归：程序调用自身的编程技巧称为递归，自己调用自己。

function factorial(num) {

return ( num <= 1 ) ? 1 ： num \* factorial(num - 1)；

}

console.log(factorial(8))；

**21、js的异步加载有哪几种方法？**

A：<script>标签的async="async"属性。HTML5中新增的属性，IE6~8 不支持 。此外，这种方法不能保证 脚本按顺序执行

B：<script>标签的defer="defer"属性。兼容所有浏览器。此外，这种方法可以确保所有设置defer属 性的 脚本按顺序执行。

C：AJAX eval（使用AJAX得到脚本内容，然后通过eval\_r(xmlhttp.responseText)来运行脚本） 。兼容 所有浏览器。

D：iframe方式 。兼容所有浏览器。

**22、列举浏览器对象模型BOM里常用的至少4个对象，并列举window对象的常用方法至少5个？**

对象：Window、 document、 location、 screen、 history、 navigator 、data

方法：Alert()、 confirm() 、prompt()、 open() 、close()

**23、简述列举文档对象模型DOM里document的常用的查找访问节点的方法做简单说明？**

document.getElementById 根据元素id查找元素

document.getElementByName 根据元素name查找元素

document.getElementTagName 根据指定的元素名查找元素

**24、原生JS的window.onload与Jquery的$(document).ready(function() {})，$(function() {})有什么不同？**

1.执行时间

window.onload必须等到页面内包括图片的所有元素加载完毕后才能执行。 $(document).ready()是DOM结构绘制完毕后就执行，不必等到加载完毕。

1. 编写个数不同

window.onload不能同时编写多个，如果有多个window.onload方法，只会执行一个 $(document).ready()可以同时编写多个，并且都可以得到执行

1. 简化写法

window.onload没有简化写法 $(document).ready(function(){})可以简写成$(function(){})；

**25、简述在jQuery中.eq()和.get()的异同？**

相同：

get() ：取得其中一个匹配的元素。数字序号表示取得第几个匹配的元素

eq()：获取第N个元素，下标都是从0开始， 用法基本相同。

不同：

eq返回的是一个jquery对象； 返回的是jQuery对象，就可以继续调用其他方法。

get返回的是一个html 对象数组；不能调用jQuery的其他方法；

**26、简述forin 循环的特点及使用场景？**

for...in 语句用于对数组或者对象的属性进行循环操作。 for ... in 循环中的代码每执行一次，就会对数组的元素或者对象的属性进行一次操作。 注意：forin循环不会按照属性的下标来排列输出。

for (变量 in 对象){

在此执行代码

}

“变量”用来指定变量，指定的变量可以是数组元素，也可以是对象的属性。

**27、****Javascript内置的常用对象有哪些？并列举该对象常用的方法？**

Arguments 函数参数集合

arguments[] 函数参数的数组

Arguments 一个函数的参数和其他属性

Arguments.callee 当前正在运行的函数

Arguments.length 传递给函数的参数的个数

Array 数组

length属性：动态获取数组长度

join()：将一个数组转成字符串。返回一个字符串。

reverse()：将数组中各元素颠倒顺序

delete运算符：只能删除数组元素的值，而所占空间还在，总长度没变(arr.length)。

shift()：删除数组中第一个元素，返回删除的那个值，并将长度减1。

pop()：删除数组中最后一个元素，返回删除的那个值，并将长度减1。

unshift()：往数组前面添加一个或多个数组元素，长度要改变。arrObj.unshift(“a”，“b，“c”)

push()：往数组结尾添加一个或多个数组元素，长度要改变。arrObj.push(“a”，“b”，“c”)

concat() 连接数组

slice() 返回数组的一部分

sort() 对数组元素进行排序，从小到大，从第一个数字开始比较排序

splice() 插入、删除或替换数组的元素

toLocaleString() 把数组转换成局部字符串

toString() 将数组转换成一个字符串

Boolean 布尔对象

Boolean.toString() 将布尔值转换成字符串

Boolean.valueOf() Boolean对象的布尔值

Date 日期时间

创建Date对象的方法

（1）创建当前(现在)日期对象的实例，不带任何参数 var today=new Date()；

（2）创建指定时间戳的日期对象实例，参数是时间戳。 时间戳：是指某一个时间距离1970年1月1日0时0分0秒，过去了多少毫秒值(1秒=1000毫秒)。 var timer=new Date(10000)； //时间是 1970年1月1日0时0分10秒

（3）指定一个字符串的日期时间信息，参数是一个日期时间字符串 var timer=new Date(“2015/5/2510：00：00”)；

（4）指定多个数值参数 var timer=new Date(2015+100，4，25，10，20，0)； //顺序为：年、月、日、时、分、秒，年、月、日是必 须的。

//获取是 get 设置改成 set就可以了

Date.getYear() 返回Date对象的年份字段(世界时)

Date.getMonth() 返回Date对象的月份字段

Date.getDate() 返回一个月中的某一天

Date.getDay() 返回一周中的某一天

Date.getHours() 返回Date对象的小时字段

Date.getMinutes() 返回Date对象的分钟字段

Date.getSeconds() 返回Date对象的秒字段

Date.getTime() 返回Date对象的毫秒表示

Date.getTimezoneOffset() 判断与GMT的时间差

Date.toDateString() 返回Date对象日期部分作为字符串

Date.toGMTString() 将Date转换为世界时字符串

Date.toLocaleDateString() 回Date对象的日期部分作为本地已格式化的字符串

Date.toLocaleString() 将Date转换为本地已格式化的字符串

Date.toLocaleTimeString() 返回Date对象的时间部分作为本地已格式化的字符串

Date.toString() 将Date转换为字符串

Date.toTimeString() 返回Date对象日期部分作为字符

.toUTCString() 将Date转换为字符串(世界时)

Date.UTC() 将Date规范转换成毫秒数

Date.valueOf() 将Date转换成毫秒表示

Error 异常对象

Error.message 可以读取的错误消息

Error.name 错误的类型

Error.toString() 把Error对象转换成字符串

EvalError 在不正确使用eval()时抛出

SyntaxError 抛出该错误用来通知语法错误

RangeError 在数字超出合法范围时抛出

ReferenceError 在读取不存在的变量时抛出

TypeError 当一个值的类型错误时，抛出该异常

URIError 由URl的编码和解码方法抛出

Function 函数构造器

Function.apply( ) 将函数作为一个对象的方法调用

Function.arguments[] 传递给函数的参数

Function.call() 将函数作为对象的方法调用

Function.caller 调用当前函数的函数

Function.length 已声明的参数的个数

Function.prototype 对象类的原型

Function.toString() 把函数转换成字符串

Math 数学对象 Math对象是一个静态对象

Math.PI：圆周率。

Math.abs()：绝对值。

Math.ceil()：向上取整(整数加1，小数去掉)。

Math.floor()：向下取整(直接去掉小数)。

Math.round()：四舍五入。

Math.pow(x，y)：求x的y次方。

Math.sqrt()：求平方根。

Number 数值对象

Number.MAX\_VALUE 最大数值

Number.MIN\_VALUE 最小数值

Number.NaN 特殊的非数字值

Number.NEGATIVE\_INFINITY 负无穷大

Number.POSITIVE\_INFINITY 正无穷大

Number.toLocaleString() 把数字转换成本地格式的字符串

Number.toPrecision() 格式化数字的有效位

Number.toString() 将—个数字转换成字符串

Number.valueOf() 返回原始数值

Object 基础对象 Object 含有所有JavaScript对象的特性的超类

Object.constructor 对象的构造函数

Object.hasOwnProperty() 检查属性是否被继承

Object.isPrototypeOf() 一个对象是否是另一个对象的原型

Object.propertyIsEnumerable() 是否可以通过for/in循环看到属性

Object.toLocaleString() 返回对象的本地字符串表示

Object.toString() 定义一个对象的字符串表示

Object.valueOf() 指定对象的原始值

RegExp 正则表达式对象

RegExp.exec() 通用的匹配模式

RegExp.global 正则表达式是否全局匹配

RegExp.ignoreCase 正则表达式是否区分大小写

RegExp.lastIndex 下次匹配的起始位置

RegExp.source 正则表达式的文本

RegExp.test() 检测一个字符串是否匹配某个模式

RegExp.toString() 把正则表达式转换成字符串

String 字符串对象

length：获取字符串的长度。如：varlen=strObj.length

toLowerCase()：将字符串中的字母转成全小写。如：strObj.toLowerCase()

toUpperCase()：将字符串中的字母转成全大写。如：strObj.toUpperCase()

charAt(index)：返回指定下标位置的一个字符。如果没有找到，则返回空字符串。

substr()：在原始字符串，返回一个子字符串 abcdefgh”.substr(2，3)= “cde” '''

substring()：在原始字符串，返回一个子字符串。 区别：''' “abcdefgh”.substring(2，3)= “c” “

split()：将一个字符串转成数组。

charCodeAt() 返回字符串中的第n个字符的代码

concat() 连接字符串

fromCharCode() 从字符编码创建—个字符串

indexOf() 返回一个子字符串在原始字符串中的索引值(从左往右查找)。如果没有找到，则返回-1。

lastIndexOf() 从后向前检索一个字符串

localeCompare() 用本地特定的顺序来比较两个字符串

match() 找到一个或多个正则表达式的匹配

replace() 替换一个与正则表达式匹配的子串

search() 检索与正则表达式相匹配的子串

slice() 抽取一个子串

toLocaleLowerCase() 把字符串转换小写

toLocaleUpperCase() 将字符串转换成大写

toLowerCase() 将字符串转换成小写

toString() 返回字符串

valueOf() 返回字符串

**28、split()join()的区别？**

split() 方法通过把字符串分割成子字符串来把一个 String 对象分割成一个字符串数组。

语法 str.split([separator][， limit])

参数 separator 指定用来分割字符串的字符（串）。separator 可以是一个字符串或正则表达式。 如果忽略 separator，则返回整个字符串的数组形式。如果 separator 是一个空字符串，则 str 将会把原字符串中每个字符的 数组形式返回。 参数limit是一个整数，限定返回的分割片段数量。split 方法仍然分割每一个匹配的 separator，但是返回的数组 只会截取最多 limit 个元素。

"|a|b|c".split("|") //将返回[""， "a"， "b"， "c"]

"hello".split("") //可返回 ["h"， "e"， "l"， "l"， "o"]

"hello".split(""， 3) //可返回 ["h"， "e"， "l"]

join() 方法将数组中的所有元素连接成一个字符串。

语法 str = arr.join([separator = '，'])

参数separator 可选，用于指定连接每个数组元素的分隔符。分隔符会被转成字符串类型；如果省略的话，默认为 一个逗号。如果 seprator 是一个空字符串，那么数组中的所有元素将被直接连接。

例子： var a = ['Wind'， 'Rain'， 'Fire']；

var myVar1 = a.join()； // myVar1的值变为"Wind，Rain，Fire"

var myVar3 = a.join(' + ')； // myVar3的值变为"Wind + Rain + Fire"

var myVar4 = a.join('')； // myVar4的值变为"WindRainFire"

**29、例举3种强制类型转换和2种隐式类型转换？**

强制 转化成字符串 toString()String()

转换成数字 Number()、 parseInt()、 parseFloat()

转换成布尔类型 Boolean()

隐式 拼接字符串 例子 var str=""+18 -/%===

**30、Html5重要的新的表单元素有哪些(至少举例5个)？**

1输入类型：部分类型是针对移动设备生效的，且具有一定的兼容性，在实际应用当中可选择性的使用。

email 输入email格式

tel 手机号码

url 只能输入url格式

number 只能输入数字

search 搜索框

range 范围，自由拖动滑块

color 拾色器

time小时

date 年月日

datetime 事件输入框

month 年月

week 年周

2表单元素（标签）

<datalist> 下拉选项，使用中文时要注意

<keygen> 生成加密字符串

<output> 不可当做数据提交？

<meter> 表示度量器，不建议用作进度条

3表单属性

placeholder 占位符

autofocus 获取焦点

multiple 文件上传多选或多个邮箱地址

autocomplete 自动完成，用于form元素，也可用于部分input，默认值on

form 指定表单项属于哪个 form，处理复杂表单时会需要

novalidate 关闭验证，可用于<form>标签， （只适应用form）

required 验证条件，必填项

pattern 正则表达式 自定义验证规则

4表单事件

oninput 用户输入内容时触发，可用于移动端输入字数统计

oninvalid 验证不通过时触发

**31、HTML5中的本地存储概念是什么？生命周期有多长？**

答：随着互联网的快速发展，基于网页的应用越来越普遍，同时也变的越来越复杂，为了满足各种各样的需 求，会经常性在本地存储大量的数据，传统方式我们以

document.cookie来进行存储的，其存储大小 只有4k左右，并且解析也相当的复杂，给开发带来诸多不便

HTML5的storage 提供了一种方式让网站把信息存储到你本地的计算机上，并在以后需要的时候进行获取。

1 特性

1. 设置、读取方便
2. 容量较大，sessionStorage约5M、localStorage约20M
3. 只能存储字符串，可以将对象JSON.stringify() 编码后存储

2 window.sessionStorage 1、生命周期为关闭浏览器窗口 2、在同一个窗口下数据可以共享

3 window.localStorage 1、永久生效，除非手动删除 2、可以多窗口共享

4相关的一些方法详解

setItem(key， value) 设置存储内容

getItem(key) 读取存储内容

removeItem(key) 删除键值为key的存储内容

clear() 清空所有存储内容

key(n) 以索引值来获取存储内容

**32、Post与get的区别，什么时候用post，什么时候用get？**

区别： get存储内容小，不能超过2kb，有限；

文件上传只能用post。

get不安全，显示在地址栏。

get效率高，因为post请求需要加密和解密的过程，get不需要。

在做数据查询时，建议用Get方式；

而在做数据添加、修改或删除时，建议用Post方式；

在提交一些不紧要信息 时，使用get，效率高。

**33、Javascript面向对象中继承实现？**

继承：就是自己没有， 别人有. 拿过来为自己所用， 并成为自己的东西

传统继承基于模板，js 继承基于对象

1. 原型链 2.借用构造函数 3.组合继承 4.原型式继承 5.寄生式继承 6.寄生组合式继承

**34、’data-’属性的作用是什么？如何获取’data-’属性的值？**

答：data-属性是H5中的，用来自定义属性，通过dataset属性获取。 不是所有的浏览器都支持，不支持用getAttribute获取。

**35、JavaScript的事件流模型都有什么，以及怎么阻止他们？**

1、原始事件模型 普通的事件绑定，比如事件赋值，按钮上绑定事件等

2、DOM2事件模型

addEventListener("eventType"，"handler"，"true!false")；

removeEventListner("eventType"，"handler"，"true!false")；

气泡模型（与ie有点区别） ，冒泡和捕获

3、IE模型 气泡模型

attachEvent("eventType"，"handler")

detachEvent("eventType"，"handler") 与dom2不同的是eventType有on前缀

**36、选其中一个正则问题作答(写出正则即可)**

a) 写一个控制在 30天到 180天的正则表达式 /^([3-9]|1[1-8])\d$/

b) 写一个能够验证 座机号码 的正则表达式。如：010-12345678 /^\d{3}\_\d{8}$/

**三、其他问题（相关的优化问题）**

**1、请谈谈你对性能优化的认识？**

1. 减少http请求次数

80%的响应时间花在下载网页内容(images， stylesheets， javascripts， scripts， flash等)。减少请求次数是缩短响应时间的关键！可以通过简化页面设计来减少请求次数，但页面内容较多可以采用以下技巧。

1. 减少DNS查询次数 （DNS域名系统）

DNS查询也消耗响应时间，如果我们的网页内容来自各个不同的domain(比如嵌入了开放广告，引用了外部 图片或脚本)，那么客户端首次解析这些domain也需要消耗一定的时间。DNS查询结果缓存在本地系统和浏览器 中一段时间，所以DNS查询一般是对首次访问响应速度有所影响。

1. 缓存Ajax

Ajax可以帮助我们异步的下载网页内容，但是有些网页内容即使是异步的，用户还是在等待它的返回结果， 例如ajax的返回是用户联系人的下拉列表。所以我们还是要注意尽量应用延迟加载提高ajax的响应速度。

1. 延迟加载

这里讨论延迟加载需要我们知道我们的网页最初加载需要的最小内容集是什么。剩下的内容就可以推到延迟 加载的集合中。 Javascript是典型的可以延迟加载内容。一个比较激进的做法是开发网页时先确保网页在没有 Javascript的时 候也可以基本工作，然后通过延迟加载脚本来完成一些高级的功能

1. **如何避免XSS？**

**XSS：Cross Site Scripting 跨站点脚本攻击，指的是攻击者利用网站程序对用户输入过滤不足，向web页面里插入攻击脚本，用户的浏览器无法识别这段脚本是不可信的所以仍会执行，从而盗用用户资料、利用用户身份进行某种动作或者对访问者进行病毒侵害的一种攻击方式**

**发生的原因：在提交表单，点击链接时对用户的输入没有进行正确的编码和过滤处理就写到数据库或html，js中**

XSS攻击分为三种，分别是：

1. Reflected XSS（基于反射的XSS攻击）

2. Stored XSS（基于存储的XSS攻击）

3. DOM-based or local XSS（基于DOM或本地的XSS攻击）

Reflected XSS

基于反射的XSS攻击，主要依靠站点服务端返回脚本，在客户端触发执行从而发起Web攻击。

开发安全措施：

1. 前端在显示服务端数据时候，不仅是标签内容需要过滤、转义，就连属性值也都可能需要。

2. 后端接收请求时，验证请求是否为攻击请求，攻击则屏蔽。

Stored XSS

基于存储的XSS攻击，是通过发表带有恶意跨域脚本的帖子/文章，从而把恶意脚本存储在服务器，每个访问该帖子/文章的人就会触发执行。

开发安全措施：

1. 首要是服务端要进行过滤，因为前端的校验可以被绕过。

2. 当服务端不校验时候，前端要以各种方式过滤里面可能的恶意脚本，例如script标签，将特殊字符转 换成HTML编码。

DOM-based or local XSS

基于DOM或本地的XSS攻击。一般是提供一个免费的wifi，但是提供免费wifi的网关会往你访问的任何页面插入一段脚本或者是直接返回一个钓鱼页面，从而植入恶意脚本。这种直接存在于页面，无须经过服务器返回就是基于本地的XSS攻击。

开发安全措施：

1. 使用HTTPS！HTTPS会在请求数据之前进行一次握手，使得客户端与服务端都有一个私钥，服务端用这个私钥加密，客户端用这个私钥解密，这样即使数据被人截取了，也是加密后的数据

防范措施

1. 输入检测，对特殊字符如<,>,',"；，等进行过滤输出检测
2. 任何内容写到页面之前都必须encode编码，对变量使用htmlEncode, js使用javascriptEncode

不要在页面中插入任何不可信数据，除非这些数已经据根据下面几个原则进行了编码

&     –>     &amp;

<     –>     &lt;

>     –>     &gt;

”     –>     &quot;

‘     –>     &#x27;

/     –>     &#x2f;

推荐使用OWASP提供的[ESAPI](https://www.owasp.org/index.php/ESAPI" \t "https://www.oschina.net/news/43919/_blank)函数库，它提供了一系列非常严格的用于进行各种安全编码的函数。

|  |
| --- |
| 插入到 html String encodedContent = ESAPI.encoder().encodeForHTML(request.getParameter(“input”)); |
| 插入到html属性String encodedContent =ESAPI.encoder().encodeForHTMLAttribute(request.getParameter(“input”)); |
| 插入到script String encodedContent = ESAPI.encoder().encodeForJavaScript(request.getParameter(“input”)); |
| 插入到style属性 String encodedContent = ESAPI.encoder().encodeForCSSL(request.getParameter(“input”)); |
| 插入到html url中 String encodedContent = ESAPI.encoder().encodeForURL(request.getParameter(“input”)); |

1. 为cookie加上HttpOnly标记防止窃取Cookie //cookie防盗

被标记的cookie是不允许任何脚本读取或修改的

1. 严格限制URL访问
2. 使用post提交表单
3. 将单步流程改为多步，在多步流程中引入校验码
4. 用户注意事项：①不要轻易打开不明连接②可以使用禁用脚本功能③在安全的浏览器上网

**3、平时如何管理项目？**

A.项目需求

明确需求

B.开发规范

变量、包、方法等的命名，代码的可读性、编写风格，公共变量的定义和使用是否符合规则，注释是否完整

C.开发与测试环境

D.代码控制

E.开发过程

**4、请谈谈项目的迭代周期？**

答：软件项目开发，一般都会采用增量、迭代、（或者叫进化、演化、演进）的软件开发模型，众多的软件开 发模型大多是以经典的瀑布模型为基础进行改进、变形，改进原则是：增加客户在整个项目周期中的参与度，降 低软件开发过程中的风险，增强软件项目的后期可维护性。 不同的软件开发模型，迭代周期长短也不相同，有的是一个月，有的是两周，我们一般都是根据实际情况确 定，一个周期完成，将项目成果（可运行的软件）提交给用户（或进行内部评审），通过后就进入下一个迭代开发周期。

**5、****工作中用过什么构建工具？**

用过gulp。

1. 全局安装 gulp：(已经安装了nodejs环境)

$ npm install --global gulp

2. 作为项目的开发依赖（devDependencies）安装：

$ npm install --save-dev gulp

3. 在项目根目录下创建一个名为 gulpfile.js 的文件：

var gulp = require('gulp');

gulp.task('default', function() {

// 将你的默认的任务代码放在这

});

1. 运行 gulp：

切换到存放gulpfile.js文件的目录，然后在命令行中执行gulp命令就行了

$ gulp

默认的名为 default 的任务（task）将会被运行，在这里，这个任务并未做任何事情。

想要单独执行特定的任务（task），请输入 gulp <task> <othertask>。

gulp API

①gulp.src() 获取文件

globs参数是文件匹配模式(类似正则表达式)，用来匹配文件路径(包括文件名)，当然这里也可以直接指定某个具体的文件路径。当有多个匹配模式时，该参数可以为一个数组。  
 options为可选参数。通常情况下我们不需要用到。

\* 匹配文件路径中的0个或多个字符，但不会匹配路径分隔符，除非路径分隔符出现在末尾

\*\* 匹配路径中的0个或多个目录及其子目录,需要单独出现，即它左右不能有其他东西了。如果出现在末尾，也能匹配文件。

? 匹配文件路径中的一个字符(不会匹配路径分隔符)

[...] 匹配方括号中出现的字符中的任意一个，当方括号中第一个字符为^或!时，则表示不匹配方括号中出现的其他字符中的任意一个，类似js正则表达式中的用法

a{0..3}d 会展开为 a0d,a1d,a2d,a3d

a{b,c{d,e}f}g 会展开为 abg,acdfg,acefg

②gulp.dest() 用来写文件

path为写入文件的路径  
 options为一个可选的参数对象，通常我们不需要用到

③gulp.task() 定义任务

gulp.task(name[,deps],fn)

name 为任务名  
 deps 是当前定义的任务需要依赖的其他任务，为一个数组。当前定义的任务会在所有依赖的任务执行完毕后才开始执行。如果没有依赖，则可省略这个参数  
 fn 为任务函数，我们把任务要执行的代码都写在里面。该参数也是可选的。

④gulp.watch() 用来监视文件的变化

gulp.watch(glob[, opts], tasks)

glob 为要监视的文件匹配模式，规则和用法与gulp.src()方法中的glob相同。  
 opts 为一个可选的配置对象，通常不需要用到  
 tasks 为文件变化后要执行的任务，为一个数组

不使用Gruntfile的原因

配置和运行分离 程序员都知道，变量的声明和使用挨在一起，最方便理解和修改。但Gruntfile里，配置Task和调用它们的地 方离得很远，极大地增加了心智负担 每个插件做的事太多 每个Task的结果必须写到磁盘文件，另一个Task再读，损害性能倒是小事，更麻烦的是让整个过程变复杂 了。就像一个个小作坊，来料加工又返回给客户，这中间的沟通成本、出错机会都大大增加。配置项过多做事多 了，配置项自然也多。至少输入和输出的位置得配吧。每个插件的配置规则还不尽相同。用每个插件，都得去学 习一番。

**6、谈谈你对****模块化的理解？**

答： 模块化就是为了减少系统耦合度，提高高内聚，减少资源循环依赖，增强系统框架设计。让开发者便于维护，同时也让逻辑相同的部分可复用

• 模块化开发：针对js、css，以功能或业务为单元组织代码。

js方面解决独立作用域、依赖管理、 api暴露、按需加载与执行、安全合并等问题，css方面解决依赖管理、组件内部样式管理等问题。

模块化的过程：

• 1、拆分 将整个系统按功能，格式，加载顺序，继承关系分割为一个一个单独的部分. 注意：拆分的粒度问题，可复用问题，效率问题.如何这些问题处理的不好，就有可能出现不想要的后果。

• 2、组合 将功能或特征相似的部分组合在一起，组成一个资源块. 将每个资源块按找需求，功能场景以及目录约束放到固定的地方以供调用.

从最开始到现在成熟方案：

1. namespace
2. sass，less
3. AMD&CMD
4. html模版
5. grunt，gulp，webpack
6. FIS，YUI，KISSY

AMD 即Asynchronous Module Definition，中文名是异步模块定义。它是一个在浏览器端模块化开发的规范

由于不是JavaScript原生支持，使用AMD规范进行页面开发需要用到对应的库函数，也就是大名鼎鼎 RequireJS

requireJS主要解决两个问题

多个js文件可能有依赖关系，被依赖的文件需要早于依赖它的文件加载到浏览器

js加载的时候浏览器会停止页面渲染，加载文件越多，页面失去响应时间越长

看一个使用requireJS的例子

// 定义模块 myModule.js

define(['dependency'], function(){

var name = 'Byron';

function printName(){

console.log(name);

}

return {

printName: printName

};

});

// 加载模块

require(['myModule'], function (my){

　 my.printName();  
 });

语法

requireJS定义了一个函数 define，它是全局变量，用来定义模块

define(id?, dependencies?, factory);

id：可选参数，用来定义模块的标识，如果没有提供该参数，脚本文件名（去掉拓展名）

dependencies：是一个当前模块依赖的模块名称数组

factory：工厂方法，模块初始化要执行的函数或对象。如果为函数，它应该只被执行一次。如果是对象，此对象应该为模块的输出值

在页面上使用require函数加载模块

require([dependencies], function(){});

参1是一个数组，表示所依赖的模块

参2是一个回调函数，当前面指定的模块都加载成功后，它将被调用。加载的模块会以参数形式传入该函数，从而在回调函数内部就可以使用这些模块

require()函数在加载依赖的函数的时候是异步加载的，这样浏览器不会失去响应，它指定的回调函数，只有前面的模块都加载成功后，才会运行，解决了依赖性的问题。

CMD 即Common Module Definition通用模块定义，CMD有个浏览器的实现SeaJS，SeaJS要解决的问题和requireJS一样，只不过在模块定义方式和模块加载（可以说运行、解析）时机上有所不同

语法

Sea.js 推崇一个模块一个文件，遵循统一的写法

define

define(id?, deps?, factory)

特点：因为CMD推崇一个文件一个模块，所以经常就用文件名作为模块id

CMD推崇依赖就近，所以一般不在define的参数中写依赖，在factory中写

function(require, exports, module)

require是一个方法，接受模块标识作为唯一参数，用来获取其他模块提供的接口

require(id)

exports 是一个对象，用来向外提供模块接口

module 是一个对象，上面存储了与当前模块相关联的一些属性和方法

demo

// 定义模块 myModule.js

define(function(require, exports, module) {

var $ = require('jquery.js')

$('div').addClass('active');

});

// 加载模块

seajs.use(['myModule.js'], function(my){});

**7、平时都用什么第三方插件？**

答：pullpage，zepto，underscore，JqueryUI，JqueryMobile，Echart，ueditor，animate.js等。

**8、请描述一下****cookie，sessionStorage和localStorage的区别？**

cookies兼容所有的浏览器，Html5 提供的storage存储方式。

1 Document.cookie

2 Window.localstorage

3 Window.sessionstorage

localstorage（本地存储），sessionstorage（会话存储）是Web Storage的两种形式

1. 存储位置不同

cookie 数据始终在同源的 http 请求中携带（即使不需要），即 cookie 在浏览器和服务器间来回传递。 sessionStorage和localStorage 不会自动把数据发给服务器，仅在本地保存。

1. 存储大小限制也不同

cookie数据不能超过4k，同时因为每次http请求都会携带cookie，所以cookie只适合保存很小的数据，如会话标识。sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可 以达到5M或更大。

1. 数据有效期不同

sessionStorage：仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持；

localStorage： 始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；

cookie只在设置的cookie过期时间之前一直有效， 即使窗口或浏览器关闭。

1. 作用域不同

sessionStorage 不在不同的浏览器窗口中共享，即使是同一个页面；

localStorage 在所有同源窗 口中都是共享的；

cookie也是在所有同源窗口中都是共享的。

1. **如何使用缓存？**

**缓存：**缓存就是将用户频繁使用的数据放在内存中以便快速访问。在用户执行一次查询操作后，查询的记录会放在缓存中。当用户再次查询时，系统会首先从缓存中读取，如果缓存中没有，再查询数据库。

使用缓存可以避免重新从读取服务器端读取数据，节省数据从客户端到服务器间往返的时间，同时也减轻了服务器数据存取的压力。

可以基于http的头信息控制缓存 ajax请求对早期的IE浏览器默认就是缓存的，可以通过时间戳防止缓存。

那么在实际项目中，通常是怎样使用缓存呢？不往大的谈，比如微博的缓存设计，这可能涉及到架构层面的东西了，单说作为一个有一定访问量的后台，又想使用缓存来提高速度的情况下。一个重要的原则（个人目前碰得多的情况）是，以传入的参数来拼键名，以这个键名来存取值。比如现在Model模型类里面有个方法：getUserInfoByUid($params)，传入包含uid字段的数组查询这个用户的信息，在这儿使用缓存，先取缓存数据，取不到则去查数据库，然后重新加入缓存，最后返回结果数据，这也是使用缓存的一个通用流程。

**10、谈谈你对预加载的理解？**

Web预加载指的是在网页全加载完成之前，在页面优先显示一些主要内容，以提高用户体验。对于一个比较庞大的网站，如果没有使用预加载技术，用户界面就会长时间显示一片空白，直到资源加载完成，页面才会显示内容，用户体验很差。 通过js预先从服务加载图片资源(动态创建Image，设置src属性)，只要浏览器把图片下载到本地， 就会被缓存，再次请求相当的src时就会优先寻找浏览器缓存，提高访问速度。

**11、缓存和预加载的区别是什么？**

缓存就是把请求过的数据缓存起来，下次请求的时候直接使用缓存内容，提高响应速度

预加载指的是提前把需要的内容加载完成，访问的时候可以明天提高响应效率，比如图片的预加载（可以提前加载一定数量的图片，当用户访问图片的时候一般只看前几张，由于是预加载好的，所以速度比较快） 。

**12、图片如何压缩？**

可以使用一些在线的图片压缩工具

优先用 png 而不是 gif

压缩 png

去掉 jpg 的 metadata

压缩 gif 动画

尝试使用 png8

避免使用 AlphaImageLoader

压缩动态生成的图像

使 favicon 更小 可缓存

使用 CSSSprites

**13、压缩文件有哪些方法？**

使用Grunt、Sass、ant压缩

压缩JavaScript的 [grunt-contrib-uglify](https://github.com/gruntjs/grunt-contrib-uglify) 插件、压缩CSS代码的 [grunt-contrib-cssmin](https://github.com/gruntjs/grunt-contrib-cssmin) 插件及压缩图片的 [grunt-contrib-imagemin](https://github.com/gruntjs/grunt-contrib-imagemin) 插件

**14、****如何区分静态页面和动态页面？**

①最简单的方法就是看后缀了，html，htm, xml的是静态，php, asp, jsp, cgi的一般是动态

②动态网页网址中有两个标志性的符号“?”和“&” （有的可能没有&） ，这个问号和&就是用来带参数的。

**15、字符串拼接和模板引擎，项目中会如何操作？模板引擎减少http请求，字符串不可变?模板引擎会不会利于SEO优化？**

简单的数据渲染，拼接字符串即可，稍微复杂的业务逻辑使用前端模板引擎，过于复杂的页面基本上使 用后台渲染的方式；模板引擎会影响SEO优化，为了解决这个问题，需要关注SEO的页面最好采用后台渲染的方式。

**16、****前台兼容性问题有哪些？**

主要是常用浏览的（前端）API差异，渲染差异，等等。

**17、你****如何对网站的文件和资源进行优化?期待的解决方案包括？**

文件合并 ：目的是减少http请求

文件最小化/文件压缩 ：减少文件下载的体积

使用CDN（内容分发网络）托管 ，降低通信距离

缓存的使用（多个域名来提供缓存）：Ajax

**19、微格式到底是做啥用？**

答：是开放的数据格式，面向的是普通用户，任何用户可以透过简单的程序读取微格式内容。而不是像Flickr、 Amazon、Google等提供特定的面向技术人员的API（一般基于XML-PRC、REST，相对复杂） 。是把语义嵌入到HTML以便有助于分离式开发而制定的一些简单约定，是兼顾人机可读性设计的数据表达方式，对Web网页进行语义注解的方法。

RSS具有微格式 的部分优点，但限制还是比较多的，比如有限的元数据（标题、描述、URL等） ，不能更好地描述语义，不太容 易与已存在的工具结合等。用微格式可以来聚合外部Blog，Flickr，YouTube，MapQuest，甚至MySpace里的内容。微格式实际就是为现有的(X)HTML元素添加元数据和其他属性，增强语义。

## 微格式是html中的，通过加属性加标签的方式，形成类似json那样的数据键值对，方便爬虫，搜索引擎等等抓取内容。常见的就是作者，时间等等一类的信息。

## 特点

为特殊应用而制定的特殊格式；

可以插入网页或XML元数据中，被收割和索引；

数据可被重用、与具体应用无关；也可自定义，完全取决于应用；

**20、****懒加载是用滚轮判断高度好还是用插件？**

懒加载(Load On Demand)是一种独特而又强大的数据获取方法,它能够在用户滚动页面的时候自动获取更多的数据,而新得到的数据不会影响原有数据的显示,同时最大程度上减少服务器端的资源耗用。

当页面被请求时，只加载可视区域的图片，其它部分的图片则不加载，只有这些图片出现在可视区域时才会动态加载这些图片，从而节约了网络带宽和提高了初次加载的速度

使用插件比较好，插件考虑的问题比较全面，仅仅通过滚轮高度判断很容易导致一些副作用（比如一次 性请求多次） 。

**21、****如何缓存整个页面，在没有网络的时候可以来回的跳转？**

使用 HTML5，通过创建 cachemanifest 文件，可以轻松地创建 web 应用的离线版本。 如需启用应用程序缓存，在文档的 <html> 标签中包含 manifest 属性。 每个指定了 manifest 的页面在用户对其访问时都会被缓存。如果未指定 manifest 属性，则页面不会被缓存 （除非在 manifest 文件中直接指定了该页面） 。 manifest 文件的建议的文件扩展名是：".appcache"。 manifest 文件需要配置正确的 MIME-type，即 "text/cache-manifest"。必须在 web 服务器上进行配置。 manifest 文件可分为三个部分： CACHEMANIFEST- 在此标题下列出的文件将在首次下载后进行缓存

NETWORK- 在此标题下列出的文件需要与服务器的连接，且不会被缓存

FALLBACK- 在此标题下列出的文件规定当页面无法访问时的回退页面（比如 404 页面）

实例 - 完整的 Manifest 文件

CACHE MANIFEST # 2012-02-21 v1.0.0 /theme.css /logo.gif /main.js

NETWORK： login.asp

FALLBACK： /html5/ /404.html

**22、CDN是啥？**

CDN的全称：是ContentDeliveryNetwork，即内容分发网络，加速的意思，那么网站CND服务就是网站加速服务。 CDN加速原理：CDN加速将网站的内容缓存在网络边缘（离用户接入网络最近的地方） ，然后在用户访问网站内容的时候，通过调度系统将用户的请求路由或者引导到离用户接入网络最近或者访问效果最佳的缓存服务器上，有该缓存服务器为用户提供内容服务；相对于直接访问源站，这种方式缩短了用户和内容之间的网络距离， 从而达到加速的效果。

CDN的特点：

1. 本地加速提高了站点(尤其含有大量图片和静态页面站点)的访问速度，并大大提高以上性质站点的稳定性

2、镜像服务 消除了不同运营商之间互联的瓶颈造成的影响，实现了跨运营商的网络加速，保证不同网络中 的用户都能得到良好的访问质量。

3、远程加速 远程访问用户根据DNS负载均衡技术 智能自动选择Cache服务器，选择最快的Cache服务器， 加快远程访问的速度

4、带宽优化 自动生成服务器的远程Mirror(镜像)cache服务器，远程用户访问时从cache服务器上读取数据， 减少远程访问的带宽、分担网络流量、减轻原站点WEB服务器负载等功能。

5、集群抗攻击 广泛分布的CDN节点加上节点之间的智能冗于机制，可以有效地预防黑客入侵以及降低各 种D.D.o.S攻击对网站的影响，同时保证较好的服务质量。

**23、****浏览器一次可以从一个域名下做多少资源？**

浏览器的并发请求数目限制是针对同一域名的，同一时间针对同一域名下的请求有一定数量限制，不同浏览器这个限制的数目不一样，超过限制数目的请求会被阻塞；目前的话，所有浏览器的并发数目一般限制在10以内。

**24、什么是****垃圾回收机制（GC）？**

找出不再使用的变量（生命周期结束的变量），释放掉其占用的内存，这个过程不是实时的，按照固定的时间间隔周期性的执行。

全局变量的声明周期一直持续到浏览器关闭页面才会清除，而局部变量只是在函数执行器存在，而在这个过程中会为局部变量在栈或者堆上分配相应的空间，来存储他们的值，然后当函数要使用这些变量的值时再取出来使用。一直到函数结束（闭包会不同）执行垃圾回收机制。

GC的优点就在于它大大简化了应用层开发的复杂度，降低了内存泄露的风险。

s的两种回收机制

A、标记清除（mark and sweep）

从语义上理解就比较好理解了，大概就是当变量进入到某个环境中的时候就把这个变量标记一下，比如标记为“进入环境”，当离开的时候就把这个变量的标记给清除掉，比如是“离开环境”。而在这后面还有标记的变量将被视为准备删除的变量。

垃圾回收器会在运行的时候给存储在内存中的所有变量加上标记，然后去掉环境中的变量以及被环境中变量所引用的变量（闭包），在这些完成之后仍存在标记的就是要删除的变量了，因为环境中的变量已经无法访问到这些变量了，然后垃圾回收器相会这些带有标记的变量机器所占空间。

B、引用计数（reference counting）

引用计数的含义是跟踪记录每个值被引用的次数，当声明一个变量并将一个引用类型的值赋给该变量时，这个时候的引用类型的值就会是引用次数+1了。如果同一个值又被赋给另外一个变量，则该值的引用次数又+1。

相反如果包含这个值的引用的变量又取得另外一个值，即被重新赋了值，那么这个值的引用就减一。当这个值的引用次数编程0时，表示没有用到这个值，这个值也无法访问，因此环境就会收回这个值所占用的内存空间回收。这样，当垃圾收集器下次再运行时，它就会释放引用次数为0的值所占用的内存。

这个看起来这么好的引用计数为啥不被别人用了呢？

因为这个过程中会出现一个循环引用的问题!

简单点来说就是一个对象小a的属性，引用了对象小b。小b对象也有一个属性引用了小a,那么小a,小b互相引用对方，也就造成了循环引用的问题啦。

**25、****image和canvas在处理图片的时候有什么区别？**

image是通过对象的形式描述图片的；

canvas通过专门的API将图片绘制在画布上。

**26、简述****移动开发的注意点，如何做好不同手机的适配，你以前的项目是怎么做的?**

1、单独做移动端项目，采用百分比布局

2、采用响应式的方式做适配

3、rem

4、flexible

**27、响应式布局的时候，轮播图使用两张不同的图片去适配大屏幕和超小屏幕，还是一张图片进行压缩适配不同终端，说明原因？**

最好使用两张不同大小的图片去适配大屏幕和超小屏幕，这样可以针对不同设备的屏幕大小，来加载响应的图片，减少超小屏幕设备的网络流量消耗，加快响应速度，同时防止图片在大屏幕下分辨率不够导致失真的问题。

**28、http和tcp有什么区别？**

TPC/IP协议是传输层协议，主要解决数据如何在网络中传输，是一种“经过三次握手”的可靠的传输方式； HTTP协议即超文本传送协议(HypertextTransferProtocol)，是应用层协议，是Web联网的基础，也是手机联 网常用的协议之一，HTTP协议是建立在TCP协议之上的一种应用。

**29、向git中添加一个文件并commit，然后push到remoteserver，请写出相关命令？**

$ git add README.md

$ git commit -m "add README.md"

$ git push origin master

**30、请把以下HTML文档翻译成MarkDown格式？**

<h3>Header</h3> <p>Hello world!<a href="https：//www.google.com">Google</a></p> <ol> <li>Number One</li> <li>Number Two</li> </ol>

### Header Hello world![Google](https：//www.google.com)

1. Number One

2. Number Two

**31、你做的页面在哪些流览器测试过?这些浏览器的内核分别是什么?**

Ie(Ie内核) 火狐（Gecko） 谷歌（webkit） opear(Presto)

**32、写出几种IE6BUG的解决方法？**

1. 双边距BUGfloat引起的 使用display：inline；

2.3像素问题 使用float引起的 使用dislpay：inline-3px

1. 超链接hover 点击后失效 使用正确的书写顺序 linkvisitedhoveractive

4.Iez-index问题 给父级添加position：relative

5.Png 透明 使用js代码 改

6.Min-height 最小高度 ！Important 解决’

7.select 在ie6下遮盖 使用iframe嵌套

8. 为什么没有办法定义1px左右的宽度容器

（IE6 默认的行高造成的，使用over：hidden，zoom：0.08 line-height：1px）

**33、图片优化**

项目中图片处理相关的优化，项目中用到的优化方案，图片大小达到多少的时候选择处理？

1. 首先了解在web开发中常见的图片有那些格式。
2. JPG 通常使用的背景图片，照片图片，商品图片等等。这一类型的图片都属于大尺寸图片或较大尺寸图片 一般使用的是这种格式。
3. PNG 这种格式的又分为两种 一种PNG-8，一种 PNG-24。

PNG-8 格式不支持半透明，也是IE6兼容的图片存储方式。

PNG-24 图片质量要求较高的半透明或全透明背景，保存成 PNG-24 更合适（为了兼容 IE 可以试用 js 插件 pngfix）一般是背景图标中使用的多。

1. GIF 这种格式显而易见的是在需要gif动画的时候使用了。

2．优化方案

 A、样式代替图片

例如：半透明、圆角、阴影、高光、渐变等。这些效果主流的浏览器都能够完美支持，而对于那些低端浏览器，我们并不会完全抛弃他们， “渐进增强”则是一个很好的解决方案。

 B、精灵图

CSSSprites，将同类型的图标或按钮等背景图合到一张大图中，减少页面请求。

 C、字体图标

IconFont，将图标做成字体文件。优点是图标支持多个尺寸，兼容所有浏览器，减少页面请求等。美中不足 的是只支持纯色的icon。SVG，对于绝大多数图案、图标等，矢量图更小，且可缩放而无需生成多套图。现在主 流浏览器都支持SVG了，所以可放心使用！ 

D、Base64 将图片转化为base64编码格式，资源内嵌于CSS或HTML中，不必单独请求。

Base64格式 data：[][；charset=][；base64]，

Base64 在CSS中的使用 .demoImg{ background-image： url("data：image/jpg；base64，/9j/4QMZRXhpZgAA SUkqAAgAAAA L....")； }

Base64 在HTML中的使用 <img width="40" height="30" src="data：image/jpg；base64，/9j/4QMZRXhpZg AASUkqAAgAAAAL...." /> 

E、图片响应式

通常图片加载都是可以通过lazy加载的形式来的，那么可以在加载的时候来判断屏幕的尺寸来达到加载大图还是小图的目的来达到优化。

**34、你知道有哪些方法可以提高网站的性能？**

（1）资源加载 CSS顶部， JS底部 CSSJS文件压缩 尽量使用图片使用精灵图，字体图标 图片加载可通过懒加载的方式。 总之就是减少资源体积减少资源请求次数。

（2）代码性能

Css：

使用CSS缩写，减少代码量；

减少查询层级：如.header.logo要好过.header.top.logo；

减少查询范围：如.header>li要好过.header li；

避免TAG标签与CLASS或ID并存：如a.top、button#submit；

删除重复的CSS；

…. Html： 减少DOM节点：加速页面渲染；

正确的闭合标签：如避免使用<div/>，浏览器会多一个将它解析成<div\></div\>的过程；

减少页面重绘。比如 给图片加上正确的宽高值：这可以减少页面重绘，

…… Js： 尽量少用全局变量；

使用事件代理绑定事件，如将事件绑定在body上进行代理；

避免频繁操作DOM节点；

减少对象查找，如a.b.c.d这种查找方式非常耗性能，尽可能把它定义在变量里；

**35、设计模式有哪些？列举你在前端开发工作中自己应用到或者了解到其他框架所用到的设计模式？**

单例、工厂、观察者、适配器、代理模式。

单例模式（Singleton）：保证一个类仅有一个实例，并提供一个访问它的全局访问点。

工厂模式（工厂方法模式）：创建产品对象的工厂接口，让子类决定将哪一个类实例化，也就是将实际创建实例对象的工作推迟到子类当中，核心工厂类不再负责具体产品的创建。

抽象工厂模式（Abstract Factory）：提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。

观察者模式（Observer）：定义对象间的一种一对多的依赖关系，以便当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动刷新。

适配器模式（Adapter）：将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的类可以一起工作。

 代理模式（Proxy）：为其他对象提供一个代理以控制对这个对象的访问。

装饰模式（Decorator）：动态地给一个对象添加一些额外的职责。就扩展功能而言，它能生成子类的方式更为灵活。

**36、请描述你熟悉的语言的垃圾回收(GC)机制，他们对循环引用是如何处理的？如何查找内存泄漏(MemoryLeak)?**

JavaScript的垃圾回收机制主要是根据数据是否还存在引用，没有引用的数据空间可能在某个时间被回收； 在java中垃圾回收机制采用对象遍历来解决循环引用；windows的任务管理器就可以查看到内存泄露。

**四、Angular、主流框架和服务器相关问题**

**1、ng-app是什么？**

ng-app指令用于告诉 AngularJS 应用当前这个元素是根元素。 所有 AngularJS 应用都必须要要一个根元素。 HTML 文档中只允许有一个ng-app 指令，如果有多个ng-app指令，则只有第一个会被使用。

**3、−g是什么？**

-g是-global的简称，全局的意思。

**4、自定义指令的类型（E，A，C，M）？**

元素（E） 、属性（A） 、类（C） 、注释（M） 。

**5、$scope和自定义指令里的scope有啥区别？**

$scope对象在AngularJS中充当数据模型的作用，也就是一般MVC框架中Model得角色.但又不完全与 通常意义上的数据模型一样，因为 $scope并不处理和操作数据，它只是建立了视图和HTML之间的桥梁，让视 图和Controller之间可以友好的通讯. 自定义指令里的scope表示指令的作用域，它有三个可选值：true、false、对象{}

**6、****Ionic中的路由？**

答：Ionic也是基于Angular的，使用的是ui-router， ui-router 的核心理念是将子视图集合抽象为一个状态机，导航意味着 状态的切换，Ionic 之所以没有使用 Angular官方的ngRoute，是回ngRoute缺少一些高级的特性，比如视图命名，视图嵌套

**7、filter？**

过滤器。

**8、ng-bind？**

ng-bind指令告诉 AngularJS 使用给定的变量或表达式的值来替换 HTML 元素的内容。 如果给定的变量或表达式修改了，指定替换的 HTML 元素也会修改。

http： //www.itcast.cn Copyright©2016 上海传智播客

**9、说一说link？**

link中可以拿到scope和controller，可以与scope进行数据绑定，与其他指令进行通信。

**10、为什么angular不推荐使用dom操作？**

Angular倡导以测试驱动开发，在的service或者controller中出现了DOM操作，那么也就意味着的测试 是无法通过的 使用Angular的其中一个好处是啥，那就是双向数据绑定，这样就能专注于处理业务逻辑，无需关系一堆堆 的DOM操作。如果在Angular的代码中还到处充斥着各种DOM操作，那为什么不直接使用jquery去开发呢。

**11、看过Angular的源码吗，它是怎么实现双向数据绑定的？**

答：angular对常用的dom事件，xhr事件等做了封装， 在里面触发进入angular的digest流程。 在digest流程里面， 会从rootscope开始遍历， 检查所有的watcher。

**12、ui-router和 ng-router区别？**

答：AngularJS的ng-route模块为控制器和视图提供了[Deep-Linking]URL ui-router的核心理念是将子视图集合抽象为一个状态机，导航意味着 状态的切换。

**13、什么是指令？**

答：指令是指示计算机执行某种操作的命令，它由一串二进制数码组成。一条指令通常由两个部分组成：操 作码+地址码。

**14、service服务三种方式是什么？**

答：angularjs 中可通过三种（$provider，$factory，$service）方式自定义服务。

**15、gulp任务都是怎么定义，怎么执行的？**

答：通过 gulp.task方法定义任务，在项目中新建gulpfile.js文件，书写代码，如： var gulp = require(‘gulp’) gulp.tasks( ‘script’ // 任务名，function(){ // 在这里写任务需要执行的代码 })； 在命令输入`gulp 任务`，可以执行所在目录gulpfile.js文件中的任务。

**16、Bootstrap中最多可以分多少列？lg、md、sm、xs这几个屏幕宽度的界限是多少？**

答：12列 .col-xs- 超小屏幕手机 (<768px) .col-sm- 小屏幕平板 (≥768px) .col-md- 中等屏幕桌面显示器 (≥992px) .col-lg- 大屏幕大桌面显示器 (≥1200px)

**17、angular中方法apply和digest区别？**

当数据出现没有经过angular但是发生改变的情况下，需要调用apply。Apply 的范围比较广，只执行一 次，但是digest针对某一元素执行多次。

**18、前端路，什么是“前端路由”?什么时候适合使用“前端路由”? “前端路由”有哪些优点和缺点?**

路由 Router 前端的路由都是通过hash来实现的，hash能兼容低版本的浏览器。 Web 服务并不会解析 hash，也就是说 # 后的内容 Web 服务都会自动忽略，但是 JavaScript 是可以通 过window.location.hash读取到的，读取到路径加以解析之后就可以响应不同路径的逻辑处理。 优点可以控制 业务逻辑 做无页面刷新 体验更好 缺点页面不刷的话无法释放内存，如果过多的操作会造成页面体验不好。

**19、ng-show/hide和ng-if的区别是什么？**

Show/hide是显示隐藏，if是是否存在某一部分。

**20、react虚拟DOM运行机制是什么？**

在React中，render执行的结果得到的并不是真正的DOM节点，结果仅仅是轻量级的JavaScript对象， 我们称之为virtualDOM。虚拟dom。

**21、react中prop和state的区别？**

需要理解的是，props是一个父组件传递给子组件的数据流，这个数据流可以一直传递到子孙组件。而 state代表的是一个组件内部自身的状态（可以是父组件、子孙组件） 。

**22、redux的原理？**

答：Redux 把一个应用程序中，所有应用模块之间需要共享访问的数据，都应该放在 State对象中。这个应 用模块可能是指 ReactComponents，也可能是你自己访问 AJAXAPI 的代理模块，具体是什么并没有一定的限制。 State以 “树形” 的方式保存应用程序的不同部分的数据。这些数据可能来自于网络调用、本地数据库查询、甚 至包括当前某个 UI 组件的临时执行状态（只要是需要被不同模块访问） 。

**23、node常用模块？**

答：httpfspathurlBufferprocess

**24、了解npm，spm，nodejs吗，请简要描述？**

答：NPM便于JavaScript开发者共享和重用代码，它可以很容易地更新你的代码；再分享。是全球最大的开 源库生态系统。 SPM是淘宝社区电商业务（xTao）为外部合作伙伴（外站）提供的一套跟踪引导成交效果数据的解决方案。 Node.js 是一个基于 ChromeV8 引擎的 JavaScript 运行环境。Node.js 使用了一个事件驱动、非阻塞式 I/O 的模型，使其轻量又高效。它使我们能够在本地运行javascript。

**25、请列举在内网的两台服务器中拷贝文件的方法？用Shell脚本解答数据库？**

答：scp-P1234-rtest\_folderuser@192.168.1.20：~ 主要就是scp命令的使用

**26、请描述你所熟悉的Web服务器框架(如Django)作为一个成熟的Web框架，需要提供哪些重要的功**

能模块？

答：提供了网站开发的常用模块：处理用户请求、操作数据库、模板渲染、配置文件管理等。

http： //www.itcast.cn Copyright©2016 上海传智播客

**27、服务器Node.js 和浏览器js的区别是什么？Node.js把js从客户端迁移了到服务端、主要做了哪些**

工作？为什么说Node.js适合做高并发的互联网应用？

答：Node采用一系列“非阻塞”库来支持事件循环的方式。本质上就是为文件系统、数据库之类的资源提供 接口。Node.js 使用事件驱动，非阻塞I/O 模型而得以轻量和高效，非常适合在分布式设备上运行数据密集型的 实时应用。

**五、网络相关问题**

**1、请解释下列术语：UrlEncode，Utf8，JSON，UTC，MD5？**

答：Urlencode：将字符串以url形式编码（在编程语言中通常都有实现该功能的内置函数或者API） 。 Utf8：是一种针对Unicode的可变长度字符编码，主要用于在网页上显示各国语言字符。

**2、请解释GET/POST的区别，以及请求参数放到url里和放到body里面的区别？**

Post与Get区别

GET请求，请求的数据会附加在URL 之后，以?分割URL和传输数据，多个参数用&连接。URL的编码格 式采用的是ASCII 编码，而不是uniclde，即是说所有的非ASCII字符都要编码之后再传输。 POST请求：POST请求会把请求的数据放置在HTTP请求包的包体中。上面的 item=bandsaw就是实际的传 输数据。 因此，GET请求的数据会暴露在地址栏中，而POST请求则不会。 传输数据的大小 在HTTP规范中，没有对URL的长度和传输的数据大小进行限制。但是在实际开发过程中，对于GET，特 定的浏览器和服务器对URL的长度有限制。因此，在使用GET请求时，传输数据会受到URL长度的限制。 对于POST，由于不是URL传值，理论上是不会受限制的，但是实际上各个服务器会规定对POST提交数据 大小进行限制，Apache、IIS都有各自的配置。 安全性 Get是Form的默认方法，安全性相对比较低。 请求参数放到url里和放到body里面的区别 首先，参数的存放位置我们无法直接指定，而是不同的请求方法参数传递的方式不同。 常用的HTTP请求主要为GET请求和POST请求两种，GET请求的参数会通过以跟随在URL后边以键值对 的方式进行传递（例：key1=a&key2=b&key3...)；而POST请求的参数会通过HEADER进行传递。考虑到安全性

http： //www.itcast.cn Copyright©2016 上海传智播客

的问题，可以确定两者都不安全，原因是HTTP请求可以被轻易抓包和截获，其中的请求参数值自然会很容易被 获取。

**3、请列举出常用的HttpHeader，Cookie是怎么实现的？**

答：Content-Length，请求、响应体的数据字节大小 Accept-Encoding，请求头，可接受的文本压缩算法，如： gzip， deflate Accept-Language，请求头，支持语言，客户端浏览器的设置，如：zh-cn，zh；q=0.8，en-us；q=0.5，en；q=0.3 User-Agent，请求头，浏览器信息， Cookie，请求头，服务器或客户端在上次设置的COOKIE，包括作用域名(.360buy.com)，过期时间，键与值。 Referer，从一个连接打开一个新页面，新页面的请求一般会加此信息，标名是从哪里跳过来的，所有的页面 的打开历史链就可被挖掘出来，有利于分析用户行为与CPS分成 Cookie在浏览器本地会有一个文件存储数据，通信的时候通过请求头和响应头传递数据

1. **长连接和短连接的区别**

HTTP的长连接和短连接本质上是TCP长连接和短连接。

短连接：浏览器和服务器每进行一次HTTP操作，就建立一次连接，当任务结束就中断连接。

长连接（Keep-Alive模式）：当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的 TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Apache）中设定这个时间。实现长连接要客户端和服务端都支持长连接。

http1.0中默认是关闭的，需要在http头加入"Connection： Keep-Alive"，才能启用Keep-Alive；

http1.1中默 认启用Keep-Alive，如果加入"Connection： close"，才关闭。

目前大部分浏览器都是用http1.1协议，也就是说默认都会发起Keep-Alive的连接请求了，所以是否能完成一个完整的Keep-Alive连接就看服务器设置情况。 启用Keep-Alive的优点 避免了建立或者重新建立连接。

**6、****从服务器考虑提高网站性能？**

答：业界常用的优化WEB页面加载速度的方法（可以分别从页面元素展现，请求连接，css，js，服务器等方面 介绍）？

对于服务器方面前端能做的工作：

使用CDN加速，使用户从离自己最近的服务器下载文件；

减少Cookie的大小，使用无cookie的域，客户端请求静态文件的时候，减少 Cookie 的反复传输对主域名的 影响； 为文件头指定Expires，使内容具有缓存性；

前端优化：DNS预解析提升页面速度 <linkrel="dns-prefetch"href="http：//hm.baidu.com"/> <linkrel="dns-prefetch"href="http：//eiv.baidu.com"/>

服务器端能做的工作： 负载均衡，分布式存储，提升服务器性能等等。

**7、****什么是Daemon进程？**

Daemon()程序是一直运行的服务端程序，又称为守护进程。通常在系统后台运行，没有控制终端，不与前台交互，Daemon程序一般作为系统服务使用。Daemon 是长时间运行的进程，通常在系统启动后就运行，在系统关 闭时才结束。一般说Daemon 程序在后台运行，是因为它没有控制终端，无法和前台的用户交互。Daemon 程序 一般都作为服务程序使用，等待客户端程序与它通信。我们也把运行的Daemon程序称作守护进程。

**8、优化一个以I/O为瓶颈的程序，以下哪些方法效果比较显著，Why?**

a) 增加CPU数目 b) 提高CPU主频 c) 增大内存的容量 d) 采用多线程 e) 采用异步I/O和多路(Multlplex)I/O f) 对每次I/O进行Bath访问(多次I/O合并一次完成)

c、e、d、f提升的效果会比较显著

c通过将数据预读取到内存中（建立内存池）的方式，提高访问时候的效率，有效减少磁盘IO读写次数

1. **什么是内存对象的序列化(Serialiization)？为什么要序列化？请描述你熟悉的网络传输序列化(Serialiization)框架或格式(Server)？**

把对象转换为字节序列的过程称为对象的序列化 序列化主要用于网络传输数据及将数据保存在硬盘上 常见的序列化以后的格式有：XML Jason ，但它们都是字符串

**六、项目相关问题**

**1、请谈下团购倒计时如何实现？**

团购倒计时页面端的效果比较好实现，主要是样式和时间的操作，重要的考虑时间要和服务器端同步，其实 这个效果也可以基于服务器端推送技术来实现。

**2、轮播图有哪几种？如何实现？**

纯CSS可以实现轮播图；JS实现轮播图；Boostrap

**3、如何实现数组去重？**

A. 最简单的可以直接利用ES5的indexOf方法。

function dupRemove(arr) {

var tmp = []； //一个新的临时数组

// 判断tmp数组中是否存在arr中第i元素，如果不存在则添加到tmp数据组。

for (var i = 0； i < arr.length； i++) {

if (tmp.indexOf(arr[i]) == -1) {

tmp.push(arr[i])；

}

}

return tmp；

}

B. 还有一种比较有意思的写法

function dupRemove (arr) {

var tmp = []；

for (var i = 0， l = arr.length； i < l； i++) {

for (var j = i + 1； j < l； j++) {

if (arr[i] === arr[j]) {

j = ++i；

}

}

tmp.push(arr[i])；

}

return r；

}

**七、程序题**

**1、vara=[]； a[0]=0；a[1]=1；a[4]=4；请问 a.length 的值是多少？a[3]的输出结果是什么？**

5 undefined

**2、vara=[5，6]； varb=a； b[0]="hello"； alert(a[0])； 请问值是多少？**

"hello"

**3、typeof(null)，typeof(undefined)，typeof(NaN)，typeof(NaN==NaN)，说出上面代码执行结果？**

object undefined number boolean

**4、functiondoSomething(){**

for(var i = 0； 4 > i； i++) { var k = 100； aMrg +=’，’ + (k + i)； } } var k = 1，aMrg = k； doSomething()； aMrg +=k； log(aMrg)；

1，100，101，102，1031

http： //www.itcast.cn Copyright©2016 上海传智播客

**5、请写出下面输出的值**

Console.log(undefined|| 1)；//值\_\_\_1\_\_ Console.log(null ||NaN)；//值\_\_NaN\_\_\_ Console.log(0&&1)；//值\_\_0\_\_\_ Console.log(0&&1|| 0)；//值\_\_0\_\_\_

**6、看下列代码，<p>标签内的文字是什么颜色的？ 红色**

<style> .classA{color： blue}； .classB{color： red}； </style> <body> <p class=”classB classA”>123</p> </body>

**7、vara=[5，6]； varb=a； b[0]=“hello”； alert(a[0])； 值是多少？**

"hello"

**8、你面前有一座高塔，这座高塔有 N(N >100)个台阶，你每次只能往前迈 1 个或者 2 个台阶，请写出程序计算总共有多少种走法？**

这个案例满足斐波那契定律 1，1，2，3，5，8，13，21， 34， 55， 89， 144

var n1 = 1； var n2 = 1； var n3 = n1 + n2；

for (var i = 3； i <= n； i++) { n3 = n1 + n2； n1 = n2；//往后推一项 n2 = n3；//往后推一项 }

console.log(n3)；

**9、请阅读下面的 CSS 代码**

#left { color： white !important； }

#container #left { color： red； }

#left { color： green !important； }

.container #left { color： blue； }

则在如下html中 <div class=”container” id=”container”> <span id=”left”>left</span> </div> #left最终color属性值为？ 绿色

**10、下面这段代码想要循环延时输出结果01234，请问输出结果是否正确，如果不正确说明为什么，并修改循环内的代码使其输出正确的结果。**

for (var i = 0； i < 5； ++i) { setTimeout (function () { console.log(i + ‘’)； }，100\*i)； } 不正确，先执行FOR 循环。for 循环完成后，在去执行setTimeout。但是这个时候I已经是 5了，所以输入 了5次5 for (var i = 0； i <5； ++i) { var a = 0； setTimeout (function () { console.log(a++)； }，100\*i)； }

**11、完成函数 showlmg()，要求能够动态根据下拉列表的选项变化，更新图片的显示**

<body>

<script type=”text/javascript”>

Function showImg (oSel) {}；

</script>

<img id=”pic” src=”img1.jpg” width=”200” height=”200”> <br />

<select id=”sel” onchange=”showImg(this)”>

<option value=”img1”>城市生活</option>

<option value=”img2”>都市早报</option>

<option value=”img3”>青山绿水</option>

</select>

</body>

var pic=document.getElementById('pic')

function showImg (oSel) {

pic.src=oSel.options[oSel.selectedIndex].value；

console.log(pic.src)；

}；

答案说明：当 select发生改变的时候调用 showImg函数，实参为this（select 对象本身），可以通过select对象的属性来 为pic的 src赋值实现图片切换

**12、完成 foo()函数的内容，要求能弹出对话框提示当前选中的是第几个单选框**

<form name=”form1” onsubmit=”retuen foo()”>

<input type=”radio” name = “radioGroup”>

<input type=”radio” name = “radioGroup”>

<input type=”radio” name = “radioGroup”>

<input type=”radio” name = “radioGroup”>

</form>

var a=document.getElementsByTagName('input')；

function foo() {

for(var i=0；i<a.length；i++){

if(a[i].checked){

alert(i+1)

}

}

}

题出的有问题， onsubmit只有在提交的时候才会触发这里面没有submit按钮， 在提交事件触发的时候遍历哪个input 表单是选中状态然后alert 粗来

**13、计算下面程序运行结果**

var msg = ‘hello’；

function great(name， attr) {

name = ‘david’；

var greating = msg + name + ‘!’；

var msg = ‘您好’；

For (var i = 0 ； i < 10；i++) {

var next = msg + ‘您的id是’ + i\*2 + i；

}

console.log(arguments[0])；

console.log(arguments[1])；

console.log(greating)；

console.log(next)；

}

geat(‘Tom’)；

答案：david //参数1

hellowworld01.html：20undefined /参数 2 未传入为未定义

hellowworld01.html：21undefineddavid! //name 虽然是参数但是参数重新赋值为david 了 msg 因为变量声明提升 所以值为 undefined

您好您的 id是189//因为 number+string=string所以 for循环最后一次声明 next=\*\*\*\*18+9

**14、下面这段 JS 输出什么，并简述为什么？**

function Foo() { var i = 0； return function () { console.log(i++)； } }

var f1 = Foo()， f2 = Foo()； f1()； f1()； f2()；

console.log(i)；

0 //f1=Foo() 相当于f1赋值为函数Foo()的返回值f1=function（）{ console.log(i++) }

1 //因为f1=了一个function所以有了作用域，f2和f1不同，不在一个内存中

0

报错 //i为Foo内部的变量全局不可访问，全局中没有i变量所以会报错

**15、请写出下面输出的值**

(1)var num = 1；

var fun = function () { console.log(num)；

//值\_\_\_undefined\_\_\_ var num = 2； console.log(num)；//值\_\_\_2\_\_\_ } fun()；

(2) var num = 1；

function fun () { console.log(num)；//值\_\_\_1\_\_\_\_ num = 2； console.log(num)；//值\_\_\_2\_\_\_\_ } fun()；

**16、写出以下程序执行的结果**

1) var a = 10； a.pro = 10； console.log(a.pro + a)；

NAN number对象不可以定义私有属性 namber+非数字和字符的值就等于NaN

1. var s = ‘hello’； s.pro = ‘world’； console.log(s.pro + s)；

//undefinedhello s位字符串，字符串不可以自定义属性，所以s.pro为undefined 字符串做加运算会 强制拼接位字符串

3) console.log(typeof fn)； function fn() {}； var fn； //function 函数提升优先于变量提升

4) var f = true； If(f === true) { var a = 10； }

function fn() { var b = 20； c = 30； } fn()； console.log(a)； //10

**17、请看如下的代码，写出结果**

var a = 5，b = 3； function test() { alert(b++)； var a = 4； alert(--a)； alert(this.a)； }

1. tese()，三次 alert()的值依次是什么？335 435 535
2. new test()，三次 alert()的值依次是什么？ 33undefined 43undefined 53undefined //this 更改了指向原来是指向 window 用了 new关键字后指向 test test木有 a属性所以为 undefined

**18、p 最后显示什么颜色。怎么让 p 的颜色变成黑色，并简要说明 css 选择器优先级关系**

#classA{color：yellow}； p.classB(color：red)；

<body> <p id=”classA” class=”classB”>123</p> </body>

//p#classA{color：black}

**19、关于正则表达式声明 6 位数字的邮编，**

var reg= /\d{6}/

**25、下面哪个属于 javascript 的布尔值 (C)**

A.1.2 B.”true”

C.false D.null

**26、请选择结果为真的表达式**null== undefined

**28、声明一个对象，给它加上 name 属性和 show方法显示其 name 值，以下代码中正确的是(D)**

E.var obj= [name ： “zhangsan” ，show ： function(){alert(name)；}]； F.Varobj={name ： “zhangsan”，show ： “alert(this.name)”}； G.Varobj={name ： “zhangsan” ，show ： function(){alert(name)；}}； H.Varobj={name ： “zhangsan” ，show ： function(){alert(this.name)；}}

**29、以下过于 Array 数组对象的说法不正确的是(C)**

I.对数组里数据的排序可以用 sort函数，如果排序效果非预期，可以给 sort函数加一个排序函数的参数 J.reverse用于对数组数据的倒序排列 K.向数组的最后位置加一个新元素，可以用 pop方法 //push吧 L.unshift方法用于向数组删除一个元素

**30、要将页面的状态显示”已经选中该文本”**

M.window.status= “已经选中该文本”

**37、写一个方法获取 url？后面的参数，并将参数对象化。**

function parseQueryString(url){

var params = {}；

var arr = url.split("?")；

if (arr.length <= 1) return params；

arr = arr[1].split("&")；

for(var i=0， l=arr.length； i<l； i++){

var a = arr[i].split("=")；

params[a[0]] = a[1]；

return params；

}

}

var url = "http：//witmax.cn/index.php?key0=0&key1=1&key2=2"；

ps = parseQueryString(url)；

console.log(ps["key1"])；

**八、非技术问题**

**1、请概述一下你上家公司中项目的具体情况(工作所使用的技术，业务流程，周期)？**

我在上家公司做的的网站pc端页面的开发与维护。 主要技术：利用div+css布局，对用户注册页面开发，利用ajax技术与后台进行数据交互，使用Validation插 件进行表单验证；使用bootstrap/zepto框架开发移动端页面，解决移动端设备的适配问题；侧边菜单栏的开发； 利用css3完成轮播图动画。

业务流程：根据需求分析，进行详细的总体设计，产生各栏目文件夹的结构图，根据美工的表现需要，设计静态网页和其它动态页面界面框架，程序员进行代码开发，做一些必要的测试，由项目组共同联调测试，发现bug， 完善一些具体的细节。 开发周期：3个月

**2、常用调试和优化工具？**

Firebug+YSlow+其它Firefox扩展 浏览器自带工具，IEDeveloperToolbar，OperaDragonfly Fiddler、HTTPAnalyzer、HttpWatch、WebDeveloper、WebAccessibilityToolbar

**3、什么叫代码部署？如何部署？**

代码部署就是把开发好的网站代码放到应用服务器上对外提供服务、部署方式根据编程语言的不同而不同，

但是大体的流程是一致的，生产环境主要是通过命令加配置文件的形式进行部署。

**4、新技术通过哪些渠道了解和学习？**

优秀的博客，githup，http：//www.daqianduan.com

**5、对于前端这个岗位，兴趣的比例占多少？**

70%

**6、前端到底工作内容是什么？和UI有什么区别？**

Web前端：主要讲UI提供的设计图，编码成静态html，实现所有特效；并负责所有交互的对接，对js要求较高

UI：主要对移动端和网页的设计

**7、你当时进公司时是以什么身份进的，实习生吗？**

**8、工作中如果出现空档期的时候，你们都在做些什么？**

答：空档期的话大家就都学习，学一些新的技术，也可以跟着js大神学习嘛，不过空档期的情况也很少，一般就 几天。

**9、平常在公司有做网页制作吗？**

答：如果实在忙的时候，我也帮制作师制作一些网页。

**10、忙的时候，会帮网页制作做到什么程度，百分之多少？**

答：一般忙的时候，我会先完成我自己的工作，像一些JS的特效和交互之后，再去帮网页制作，大概是10%。

**11、你在你做过的哪个项目调试中，遇到了哪些比较深刻的部分，说一说。你发现到解决这个问题用了**

**多久？**

答：通过你描述的问题难度，和你发现到解决问题的时间，看你js程度。

**12、身为一位web前端工程师，你肯定知道现在最流行的前端技术有哪些吧？请例举3例？**

答：浏览器兼容性，hack技术，Node.js，Angular.js，Vue，react等。

**获取当前页面的url地址，并把url地址中？后面的参数变成对象取出来**

function query(url) {

var obj = {};

var str = url.split('?')[1]; //id=2&name=zs

str.split('&).forEach(function(item){ //item--> id=2

var arr = item.split('=');

obj[arr[0]] = arr[1] ? arr[1] : ' ';

})

return obj;

}

query(location.search); //http://127.0.0.1/12-detal.html?id=2&name=zs

**软件开发流程（软件开发生命周期）**

**生命周期：**孕育、诞生、成长、成熟、结束

**第一步：可行性分析与项目开发计划**

这个阶段主要确定软件开发的目标及其可行性，明确要解决的问题及解决办法，以及解决问题需要的费用、资源、时间。要进行问题定义、可行性分析，制定项目开发计划。《**项目开发计划**》

**第二步：需求分析（设计师）**

需求分析是明确软件系统要做什么，确定软件系统的功能、性能、数据、和界面等要求。《**软件需求说明书**》

**第三步：设计**

**概要设计**就是设计软件的结构，明确软件系统由那些模块组成，这些模块的层次结构、调用关系以及模块的功能，同时确定数据结构和数据库结构。

**详细设计**是对每个模块完成的功能进行具体的描述，把功能描述转变为精确地、结构化的过程描述，既该模块的控制结构或者说逻辑结构。

**《概要设计说明书》、《数据库设计说明书》、《接口设计》、《详细设计说明书》**

**第四步：编码**

编码就是把模块的控制结构转化为程序代码，该阶段需要编码规范。

开发者根据《软件系统详细设计报告》中对数据结构、算法分析和模块实现等方面的设计要求，开始具体的编写程序工作，分别实现各模块的功能，从而实现对目标系统的功能、性能、接口、界面等方面的要求。

**第五步：测试**

测试编写好的系统。《软件测试计划》、《测试用例》、《软件测试报告 》

**第六步：软件交付准备**

在软件测试证明软件达到要求后，软件开发者应向用户提交开发的目标安装程序、数据库的数据字典、《用户安装手册》、《用户使用指南》、需求报告、设计报告、测试报告等双方合同约定的产物。

**第七步：验收**

用户验收。

**第八步：软件维护**

\* 软件数据库管理

\* 用户跟踪培训

\* 故障分析解决

**第九步：软件升级**

\* 需求调整分析

\* 软件功能拓展

\* 系统优化

**第十步：报废处置**

\* 软件不能适应业务发展

\* 新软件项目立项

\* 企业数据详细备份

**[前端开发](http://lib.csdn.net/base/jquery" \o "jQuery知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)**面试知识点大纲：

**HTML&CSS：**   
 对Web标准的理解、浏览器内核差异、兼容性、hack、CSS基本功：布局、盒子模型、选择器优先级及使用、 **[HTML5](http://lib.csdn.net/base/html5" \o "HTML5知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)**、**[CSS3](http://lib.csdn.net/base/css3" \o "CSS3知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)**、移动端适应

**[JavaScript](http://lib.csdn.net/base/javascript" \o "JavaScript知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)：**   
 数据类型、面向对象、继承、闭包、插件、作用域、跨域、原型链、模块化、自定义事件、内存泄漏、事件机 制、异步装载回调、模板引擎、Nodejs、JSON、ajax等。

**其他：**   
 HTTP、安全、正则、优化、重构、响应式、移动端、团队协作、可维护、SEO、UED、**[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)**、职业生涯

作为一名前端工程师，无论工作年头长短都应该必须掌握的**知识点**：

一、DOM结构 —— 两个节点之间可能存在哪些关系以及如何在节点之间任意移动。   
二、DOM操作 ——如何添加、移除、移动、复制、创建和查找节点等。   
三、事件 —— 如何使用事件，以及IE和标准DOM事件模型之间存在的差别。   
四、XMLHttpRequest —— 这是什么、怎样完整地执行一次GET请求、怎样检测错误。   
五、严格模式与混杂模式 —— 如何触发这两种模式，区分它们有何意义。   
六、盒模型 —— 外边距、内边距和边框之间的关系，及IE8以下版本的浏览器中的盒模型   
七、块级元素与行内元素 —— 怎么用CSS控制它们、以及如何合理的使用它们   
八、浮动元素——怎么使用它们、它们有什么问题以及怎么解决这些问题。   
九、HTML与XHTML——二者有什么区别，你觉得应该使用哪一个并说出理由。   
1十、JSON —— 作用、用途、设计结构。

1. **简单介绍javascript的同源策略。**

**同源策略**

同源是指：协议，域名，端口相同。   
在JavaScript中，有一个很重要的安全性限制，被称为“Same-Origin Policy”（同源策略）。同源策略是客户端脚本(尤其是JavaScript)的重要安全度量标准。这一策略对于JavaScript代码能够访问的页面内容做了很重要的限制，即JavaScript只能访问与包含它的文档在同一域下的内容。

同源策略是一种安全协议。   
具体表现在JS中：一段脚本只能读取来自同一来源的窗口的文档和属性。(iframe.contentDocument 等访问错误)

同源策略是为了保护dom节点的安全，任何一种跨越同源策略都需要双方的配合

跨域策略

1、双方修改document.domaim来跨域,domain为主机名也就是所谓的域名

2、使用window.name

3、jsonp

4、创建一个代理

5、使用HTML5中新引进的window.postMessage方法

**2.引入样式表的方式**

在HTML中引入CSS的方法主要有四种，它们分别是行内式、内嵌式、链接式和导入式。

1.行内式   
 行内式是在标记的style属性中设定CSS样式。这种方式没有体现出CSS的优势，不推荐使用。

<div style="color:red"></div>

2.**[嵌入式](http://lib.csdn.net/base/embeddeddevelopment" \o "嵌入式开发知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)**   
 嵌入式是将CSS样式集中写在网页的<head></head>标签对的<style></style>标签对中。缺点是对于一个 包含很多网页的网站，在每个网页中使用嵌入式，进行修改样式时非常麻烦。单一网页可以考虑使用嵌入式。

<head>

<style type="text/css">

...此处写CSS样式

</style>

</head>

3.导入式   
 将一个独立的.css文件引入HTML文件中，导入式使用CSS规则引入外部CSS文件，<style>标记也是写在<head>标记中，导入式会在整个网页装载完后再装载CSS文件，因此这就导致了一个问题，如果网页比较大则会出现先显示无样式的页面，闪烁一下之后，再出现网页的样式。这是导入式固有的一个缺陷。

<style type="text/css">

@import"mystyle.css"; 此处要注意.css文件的路径

</style>

4.链接式   
 也是将一个.css文件引入到HTML文件中，但它与导入式不同的是链接式使用HTML规则引入外部CSS文件，它在网页的<head></head>标签对中使用<link>标记来引入外部样式表文件，使用链接式与导入式不同的是它会在网页文件主体装载前装载CSS文件，因此显示出来的网页从一开始就是带样式效果的，它不会象导入式那样先显示无样式的网页，然后再显示有样式的网页，这是链接式的优点。

<link href="mystyle.css" rel="stylesheet" type="text/css"/>

总结：一般来说，做网站时把样式写在多个样式表文件中，先用链接式引入一个总的CSS文件，然后在这个CSS文件中在使用导入式来引入其他的CSS文件。但如果通过JavaScrip来动态引入CSS文件则只能使用链接式。

**3.闭包是什么，有什么特性，对页面有什么影响？**

闭包是一个封闭的隔离的空间，在js中表现为函数所构成的外部无法访问的区域，当函数被外部引用时，就创建了该函数的闭包   
 闭包的实质是一个函数，是一个用于返回局部变量值的函数，因为在全局中，受JavaScript链式作用域结构的影响，父级变量中无法访问到子级的变量值，为了解决这个问题，才使用闭包这个概念。

闭包的特性：

①.封闭性：外界无法访问闭包内部的数据，如果在闭包内声明变量，外界是无法访问的，除非闭包主动向外界提供访问接口；  
 ②.持久性：一般的函数，调用完毕之后，系统自动注销函数，而对于闭包来说，在外部函数被调用之后，闭包结构依然保存在  
系统中，闭包中的数据依然存在，从而实现对数据的持久使用。

优点：

① 减少全局变量。

② 减少传递函数的参数量

③ 封装；  
   缺点：  
 使用闭包会占有内存资源，过多的使用闭包会导致内存泄露等.

function say667() {

// Local variable that ends up within closure

var num = 666;

var sayAlert = function() { alert(num); }

num++;

return sayAlert;

}

var sayAlert = say667();

sayAlert()//执行结果应该弹出的667

执行say667()后,say667()闭包内部变量会存在,而闭包内部函数的内部变量不会存在.使得Javascript的垃圾回收机制GC不会收回say667()所占用的资源，因为say667()的内部函数的执行需要依赖say667()中的变量。这是对闭包作用的非常直白的描述。

1. **ajax请求时，get和post方式的区别？怎么实现跨域请求？**

Q1:

一个在url后面 一个放在请求体里面

有大小限制

安全问题

应用不同

GET：一般用于信息获取。

使用url传递参数

对所发送信息的数量也有限制，一般在2000个字符左右。

该方法发送的信息对任何人都是可见的。所有的变量名和值都显示在url中。也就是说get请求是用url 来传递参数的。

GET请求是幂等的，请求一次和请求一万次的效果是一样的。不会因为查询次数的多少而改变数据信息。

POST：一般用于修改服务器上的资源。

对所发送信息的数量无限制。

一般用来从表单进行发送数据。数据不在url中显示。所有的变量和值对别人不可见，而是嵌入到请求体中。

Q2:   
 处理跨域方法一：代理   
 通过同域名的web服务器端创建一个代理：   
 处理跨域方法二——JSONP   
 JSONP可用于解决主流浏览器的跨域数据get请求访问的问题。   
 jQuery中的ajax方法是支持jsonp的处理方式。   
 前端只需要修改dataType:”jsonp”   
 同时在其下一行添加jsonp:”callback”

**5、请简单说明怎么样实现与后台交互？**

1. **前后端沟通设计接口**
2. **前端通过ajax进行网络请求，获取服务器返回的数据**

**6.什么是HTTP协议？**

A、HTTP协议（HyperText Transfer Protocol，超文本传输协议）是因特网上应用最为广泛的一种网络传输协议，所有的WWW文件都必须遵守这个标准。   
 B、HTTP是一个基于TCP/IP通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。   
 C、HTTP是在网络上传输HTML的协议，用于浏览器和服务器的通信。   
 D、HTTP协议工作于客户端-服务端架构为上。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。

**10.ajax请求时，如何解析json数据？json格式如何在js中解析？**

在js中解析json有两种方式：eval和JSON.parse。 鉴于安全性考虑，使用parse更靠谱。   
 eval不仅解析了json字符串，还执行了json字符串中的js方法。eval不会判断json字符串是否合法，JSON.parse则会判断。所以使用eval方式是非常危险的，一般使用JSON.parse方式。

Angular 与**[React](http://lib.csdn.net/base/react" \o "React知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)** 有什么区别？

****HTML****

****1 . Doctype作用？严格模式与混杂模式如何区分？它们有何意义?****

（1）、位于文档中的最前面，处于 <html>标签之前。告知浏览器的解析器，用什么文档类型规范来解析这个文档。   
（2）、严格模式的排版和 JS 运作模式是以该浏览器支持的最高标准运行。   
（3）、在混杂模式中，页面以宽松的向后兼容的方式显示。模拟老式浏览器的行为以防止站点无法工作。   
（4）、DOCTYPE不存在或格式不正确会导致文档以混杂模式呈现。

****2 . HTML5 为什么只需要写 <!DOCTYPE HTML>？****

****3 . 行内元素有哪些？块级元素有哪些？ 空(void)元素有那些？****

（1）CSS规范规定，每个元素都有display属性，确定该元素的类型，每个元素都有默认的display值，比如div默认display属性值为“block”，成为“块级”元素；span默认display属性值为“inline”，是“行内”元素。   
（2）行内元素有：a b span img input select strong i   
块级元素有：div ul ol li dl dt dd h1 h2 h3 h4…p ul form   
（3）知名的空元素： <br> <hr> <img> <input> <link> <meta>   
鲜为人知的是： <area> <base> <col> <command> <embed> <keygen> <param> <source> <track> <wbr>

**4 . 页面导入样式时，使用link和@import有什么区别？**

（1）link属于XHTML标签，而@import是CSS提供的;   
（2）页面被加载的时，link会同时被加载，而@import引用的CSS会等到页面被加载完再加载;   
（3）import只在IE5以上才能识别，CSS2.1以下浏览器不支持，而link是XHTML标签，无兼容问题;   
（4）link方式的样式的权重 高于@import的权重.   
（5） Link 支持使用javascript改变样式，后者不可

****5 . 介绍一下你对浏览器内核的理解？常见的浏览器内核有哪些？**** IE浏览器的内核Trident、Mozilla的Gecko、Chrome的Blink（WebKit的分支）、Opera内核原为Presto，现为Blink；

**6 . 常见兼容性问题？**\* png24位的图片在iE6浏览器上出现背景，解决方案是做成PNG8.   
\* 浏览器默认的margin和padding不同。解决方案是加一个全局的\*{margin:0;padding:0;}来统一。   
\* IE6双边距bug:块属性标签float后，又有横行的margin情况下，在ie6显示margin比设置的大。   
浮动ie产生的双倍距离 #box{ float:left; width:10px; margin:0 0 0 100px;}   
这种情况之下IE会产生20px的距离，解决方案是在float的标签样式控制中加入 ——\_display:inline;将其转化为行内属性。(\_这个符号只有ie6会识别)   
渐进识别的方式，从总体中逐渐排除局部。   
首先，巧妙的使用“\9”这一标记，将IE游览器从所有情况中分离出来。   
接着，再次使用“+”将IE8和IE7、IE6分离开来，这样IE8已经独立识别。

css

.bb{

background-color:#f1ee18;/\*所有识别\*/

.background-color:#00deff\9; /\*IE6、7、8识别\*/

+background-color:#a200ff;/\*IE6、7识别\*/

\_background-color:#1e0bd1;/\*IE6识别\*/

}

* IE下,可以使用获取常规属性的方法来获取自定义属性,也可以使用getAttribute()获取自定义属性;Firefox下,只能使用getAttribute()获取自定义属性. 解决方法:统一通过getAttribute()获取自定义属性.
* IE下,even对象有x,y属性,但是没有pageX,pageY属性;   
  Firefox下,event对象有pageX,pageY属性,但是没有x,y属性.
* 解决方法：（条件注释）缺点是在IE浏览器下可能会增加额外的HTTP请求数。
* Chrome 中文界面下默认会将小于 12px 的文本强制按照 12px 显示, 可通过加入 CSS 属性 -webkit-text-size-adjust: none; 解决.   
  超链接访问过后hover样式就不出现了 被点击访问过的超链接样式不在具有hover和active了解决方法是改变CSS属性的排列顺序: L-V-H-A : a:link {} a:visited {} a:hover {} a:active {}

**写出几种IE6 BUG的解决方法** 1.双边距BUG float引起的 使用display   
 2.3像素问题 使用float引起的 使用dislpay:inline -3px   
 3.超链接hover 点击后失效 使用正确的书写顺序 link visited hover active   
 4.Ie z-index问题 给父级添加position:relative   
 5.Png 透明 使用js代码 改   
 6.Min-height 最小高度 ！Important 解决’   
 7.select 在ie6下遮盖 使用iframe嵌套   
 8.为什么没有办法定义1px左右的宽度容器

IE6默认的行高造成用了  over:hidden,zoom:0.08line-height:1px

**7 . html5有哪些新特性、移除了那些元素？如何处理HTML5新标签的浏览器兼容问题？ 如何区分 HTML 和 HTML5？**

* HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加。
* 绘画 canvas   
  用于媒介回放的 video 和 audio 元素   
  本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失；   
  sessionStorage 的数据在浏览器关闭后自动删除   
  语意化更好的内容元素，比如 article、footer、header、nav、section   
  表单控件，calendar、date、time、email、url、search   
  新的技术webworker, websockt, Geolocation
* 移除的元素-纯表现的元素：basefont，big，center，font, s，strike，tt，u；   
  对可用性产生负面影响的元素：frame，frameset，noframes；

**8 . 支持HTML5新标签：**

* IE8/IE7/IE6支持通过document.createElement方法产生的标签，可以利用这一特性让这些浏览器支持HTML5新标签，浏览器支持新标签后，还需要添加标签默认的样式：
* 当然最好的方式是直接使用成熟的框架、使用最多的是html5shim框架

<!--[if lt IE 9]>

<script> src="http://html5shim.googlecode.com/svn/trunk/html5.js"</script>

<![endif]--

**9 . 简述一下你对HTML语义化的理解？如何区分： DOCTYPE声明\新增的结构元素\功能元素，语义化的理解？**

**用正确的标签做正确的事情。**　　1. html语义化让页面的内容结构化，结构更清晰，便于对浏览器、搜索引擎解析;   
　　2. 即使在没有样式CSS情况下也以一种文档格式显示，并且是容易阅读的; 搜索引擎的爬虫也依赖于HTML标记来确定上下文和各个关键字的权重，利于SEO;   
　　3. 使阅读源代码的人对网站更容易将网站分块，便于阅读维护理解。

**10 . HTML5的离线储存怎么使用，工作原理能不能解释一下？**

localStorage 长期存储数据，浏览器关闭后数据不丢失；   
 sessionStorage 数据在浏览器关闭后自动删除。

**12 . 请描述一下 cookies，sessionStorage 和 localStorage 的区别？**

cookie在浏览器和服务器间来回传递。   
 sessionStorage和localStorage不会。   
 sessionStorage和localStorage的存储空间更大；   
 sessionStorage和localStorage有更多丰富易用的接口；   
 sessionStorage和localStorage各自独立的存储空间；

**13 . iframe有那些缺点？**

\*iframe会阻塞主页面的Onload事件；   
 \*iframe和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载。   
使用iframe之前需要考虑这两个缺点。如果需要使用iframe，最好是通过javascript动态给iframe添加src属性值，这样可以可以绕开以上两个问题。

**CSS**

**1 . 介绍一下标准的CSS的盒子模型？与低版本IE的盒子模型有什么不同的？**

（1）有两种：IE 盒子模型、标准 W3C 盒子模型；IE的content部分包含了 border 和 padding;   
（2）盒模型： 内容(content)、填充(padding)、边界(margin)、 边框(border).

**2 . CSS选择符有哪些？哪些属性可以继承？**

* 1.id选择器（ # myid）   
  2.类选择器（.myclassname）   
  3.标签选择器（div, h1, p）   
  4.相邻选择器（h1 + p）   
  5.子选择器（ul > li）   
  6.后代选择器（li a）   
  7.通配符选择器（ \* ）   
  8.属性选择器（a[rel = “external”]）   
  9.伪类选择器（a: hover, li: nth - child）
* 不可继承的样式：border padding margin width height ;
* 可以同父元素继承的样式：   
  　　所有元素可继承：visibility和cursor。   
  　　内联元素可继承：letter-spacing、word-spacing、white-space、line-height、color、font、 font-family、font-size、font-style、font-variant、font-weight、text- decoration、text-transform、direction。   
  　　块状元素可继承：text-indent和text-align。   
  　　列表元素可继承：list-style、list-style-type、list-style-position、list-style-image。   
  　　表格元素可继承：border-collapse。

**3 . CSS优先级[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)如何计算？**

* 优先级就近原则，同权重情况下样式定义最近者为准;
* 载入样式以最后载入的定位为准;
* 继承不如指定 Id>class>标签选择   
  优先级为:   
  !important > id > class > tag   
  important 比 内联优先级高

**5 . 如何居中div？如何居中一个浮动元素？如何让绝对定位的div居中？**Q1:   
 给div设置一个宽度，然后添加margin:0 auto属性   
 div{width:200px; margin:0 auto; }   
Q2:   
 确定容器的宽高 宽500 高 300 的层   
 设置层的外边距

.div {

Width:500px ; height:300px;//高度可以不设

margin-top:-150px

margin-left:-250px

position:absolute;//相对定位

left:50%;

top:50%;

}

Q3:

.wrapper{position:relative;} //父元素

.content{

background-color:#6699FF; //子元素

width:200px;

height:200px;

position: absolute; //父元素需要相对定位

top: 50%;

left: 50%;

margin-top:-100px ; //二分之一的height，width

margin-left: -100px;

}

**6 . display有哪些值？说明他们的作用。**

block 像块类型元素一样显示。   
 none 缺省值。像行内元素类型一样显示。   
 inline-block 像行内元素一样显示，但其内容像块类型元素一样显示。   
 list-item 像块类型元素一样显示，并添加样式列表标记。

**7 . position的值relative和absolute定位原点是？**

\*absolute 生成绝对定位的元素，相对于 static 定位以外的第一个父元素进行定位。   
 \*fixed （老IE不支持）生成绝对定位的元素，相对于浏览器窗口进行定位。   
 \*relative生成相对定位的元素，相对于其正常位置进行定位。   
 \* static 默认值。没有定位，元素出现在正常的流中   
 \*（忽略 top, bottom, left, right z-index 声明）   
 \* inherit 规定从父元素继承 position 属性的值。

**15 . 为什么要初始化CSS样式。**

因为浏览器的兼容问题，不同浏览器对有些标签的默认值是不同的，如果没对CSS初始化往往会出现浏览器之间的页面显示差异。 当然，初始化样式会对SEO有一定的影响，但鱼和熊掌不可兼得，但力求影响最小的情况下初始化。   
 \*最简单的初始化方法就是：\* {padding: 0; margin: 0;} （不建议）   
 淘宝的样式初始化：

body, h1, h2, h3, h4, h5, h6, hr, p, blockquote, dl, dt, dd, ul, ol, li, pre, form, fieldset, legend, button, input, textarea, th, td { margin:0; padding:0; }body, button, input, select, textarea { font:12px/1.5tahoma, arial, \5b8b\4f53; }h1, h2, h3, h4, h5, h6{ font-size:100%; }address, cite, dfn, em, var { font-style:normal; }code, kbd, pre, samp { font-family:couriernew, courier, monospace; }small{ font-size:12px; }ul, ol { list-style:none; }a { text-decoration:none; }a:hover { text-decoration:underline; }sup { vertical-align:text-top; }sub{ vertical-align:text-bottom; }legend { color:#000; }fieldset, img { border:0; }button, input, select, textarea { font-size:100%; }table { border-collapse:collapse; border-spacing:0; }

**21 . 请解释一下为什么会出现浮动和什么时候需要清除浮动？清除浮动的方式？** 1.使用空标签清除浮动 clear:both（理论上能清楚任何标签，增加无意义的标签）   
 2.使用overflow:auto（空标签元素清除浮动而不得不增加无意代码的弊端,,使用zoom:1用于兼容IE）   
 3.是用afert伪元素清除浮动(用于非IE浏览器)

**41 . 如果需要手动写动画，你认为最小时间间隔是多久，为什么？（阿里）**

多数显示器默认频率是60Hz，即1秒刷新60次，所以理论上最小间隔为1/60＊1000ms ＝ 16.7ms

**42 . display:inline-block 什么时候会显示间隙？(携程)**

移除空格、使用margin负值、使用font-size:0、letter-spacing、word-spacing.

**JavaScript**

**1 . 介绍JavaScript的基本数据类型。**

number,string,boolean,object,undefined

**8 . 谈谈This对象的理解。**

this是js的一个关键字，随着函数使用场合不同，this的值会发生变化。但是有一个总原则，那就是this指的是调用函数的那个对象。this一般情况下：是全局对象Global。 作为方法调用，那么this就是指这个对象

**9 . eval是做什么的？**

它的功能是把对应的字符串解析成JS代码并运行；应该避免使用eval，不安全，非常耗性能（2次，一次解析成js语句，一次执行）。

**12 . 写一个通用的事件侦听器函数(机试题)。**

// event(事件)工具集，来源：github.com/markyun

markyun.Event = {

// 页面加载完成后

readyEvent : function(fn) {

if (fn==null) {

fn=document;

}

var oldonload = window.onload;

if (typeof window.onload != 'function') {

window.onload = fn;

} else {

window.onload = function() {

oldonload();

fn();

};

}

},

// 视能力分别使用dom0||dom2||IE方式 来绑定事件

// 参数： 操作的元素,事件名称 ,事件处理程序

addEvent : function(element, type, handler) {

if (element.addEventListener) {

//事件类型、需要执行的函数、是否捕捉

element.addEventListener(type, handler, false);

} else if (element.attachEvent) {

element.attachEvent('on' + type, function() {

handler.call(element);

});

} else {

element['on' + type] = handler;

}

},

// 移除事件

removeEvent : function(element, type, handler) {

if (element.removeEnentListener) {

element.removeEnentListener(type, handler, false);

} else if (element.datachEvent) {

element.detachEvent('on' + type, handler);

} else {

element['on' + type] = null;

}

},

// 阻止事件 (主要是事件冒泡，因为IE不支持事件捕获)

stopPropagation : function(ev) {

if (ev.stopPropagation) {

ev.stopPropagation();

} else {

ev.cancelBubble = true;

}

},

// 取消事件的默认行为

preventDefault : function(event) {

if (event.preventDefault) {

event.preventDefault();

} else {

event.returnValue = false;

}

},

// 获取事件目标

getTarget : function(event) {

return event.target || event.srcElement;

},

// 获取event对象的引用，取到事件的所有信息，确保随时能使用event；

getEvent : function(e) {

var ev = e || window.event;

if (!ev) {

var c = this.getEvent.caller;

while (c) {

ev = c.arguments[0];

if (ev && Event == ev.constructor) {

break;

}

c = c.caller;

}

}

return ev;

}

};

**13 . [“1”, “2”, “3”].map(parseInt) 答案是多少？**

[1, NaN, NaN] 因为 parseInt 需要两个参数 (val, radix)，其中 radix 表示解析时用的基数。map 传了 3 个 (element, index, array)，对应的 radix 不合法导致解析失败。

**14 . 关于事件，IE与火狐的事件机制有什么区别？ 如何阻止冒泡？**

Q1. 我们在网页中的某个操作（有的操作对应多个事件）。例如：当我们点击一个按钮就会产生一个事件。是可以被 JavaScript 侦测到的行为。   
 Q2. 事件处理机制：IE是事件冒泡、火狐是 事件捕获；   
 Q3. ev.stopPropagation();

**17 . 如何判断一个对象是否属于某个类？**

使用instanceof （待完善）

if(a instanceof Person){

alert('yes');

}

**18 . new操作符具体干了什么呢?**

1、创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。   
 2、属性和方法被加入到 this 引用的对象中。   
 3、新创建的对象由 this 所引用，并且最后隐式的返回 this 。

**20 . Javascript中，有一个函数，执行时对象查找时，永远不会去查找原型，这个函数是？**

hasOwnProperty

**21 . 对JSON的了解？**

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 {'age':'12', 'name':'back'}

**23 . js延迟加载的方式有哪些？**

defer和async、动态创建DOM方式（用得最多）、按需异步载入js

**34 . 异步加载的方式有哪些？**

(1) defer，只支持IE   
 (2) async：   
 (3) 创建script，插入到DOM中，加载完毕后callBack

**35 . documen.write和 innerHTML的区别?**

document.write只能重绘整个页面   
 innerHTML可以重绘页面的一部分

**36 . DOM操作——怎样添加、移除、移动、复制、创建和查找节点?**

（1）创建新节点

createDocumentFragment() //创建一个DOM片段

createElement() //创建一个具体的元素

createTextNode() //创建一个文本节点

（2）添加、移除、替换、插入、复制

appendChild() //添加

removeChild() //移除

replaceChild() //替换

insertBefore() //在已有的子节点前插入一个新的子节点

cloneNode() //复制

（3）查找

getElementsByTagName() //通过标签名称

getElementsByName()//通过元素的Name属性的值(IE容错能力较强，会得到一个数组，其中包括id等于 name值的)

getElementById() //通过元素Id，唯一性

**37 . .call() 和 .apply() 的含义和区别？**

call() 和 apply() 是预定义的函数方法。两个方法可用于调用函数，两个方法的第一个参数必须是对象本身。 后面的参数都是传递给当前对象的参数。

  区别：apply传入的是一个参数数组，也就是将多个参数组合成为一个数组传入，而call则作为call的参数传入（从第二个参数开始）。   
 Object.call(this,obj1,obj2,obj3)   
 Object.apply(this,arguments)   
 在 JavaScript 严格模式(strict mode)下, 在调用函数时第一个参数会成为 this 的值， 即使该参数不是一个对象。 在 JavaScript 非严格模式(non-strict mode)下, 如果第一个参数的值是 null 或 undefined, 它将使用全局对象替代。

**45 . jquery中如何将数组转化为json字符串，然后再转化回来？**

jQuery中没有提供这个功能，所以你需要先编写两个jQuery的扩展：

$.fn.stringifyArray = function(array) {

return JSON.stringify(array)

}

$.fn.parseArray = function(array) {

return JSON.parse(array)

}

然后调用：

$("").stringifyArray(array)

**53 . 针对 jQuery性能的优化方法？**

\*基于Class的选择性的性能相对于Id选择器开销很大，因为需遍历所有DOM元素。   
\*频繁操作的DOM，先缓存起来再操作。用Jquery的链式调用更好。   
\*for (var i = size; i < arr.length; i++) {}   
 for 循环每一次循环都查找了数组 (arr) 的.length 属性，在开始循环的时候设置一个变量来存储这个数字，可以让循环跑得更快： for (var i = size, length = arr.length; i < length; i++) {}

**54 . Jquery与jQuery UI有啥区别？**

\*jQuery是一个js库，主要提供的功能是选择器，属性修改和事件绑定等等。   
 \*jQuery UI则是在jQuery的基础上，利用jQuery的扩展性，设计的插件。 提供了一些常用的界面元素，诸如对话框、拖动行为、改变大小行为等等

**11 . 对Node的优点和缺点提出了自己的看法？**

\*（优点）因为Node是基于事件驱动和无阻塞的，所以非常适合处理并发请求，因此构建在Node上的代理服务器相比其他技术实现（如Ruby）的服务器表现要好得多。   
此外，与Node代理服务器交互的客户端代码是由javascript语言编写的，因此客户端和服务器端都用同一种语言编写，这是非常美妙的事情。

\*（缺点）Node是一个相对新的开源项目，所以不太稳定，它总是一直在变，而且缺少足够多的第三方库支持。看起来，就像是Ruby/Rails当年的样子。

**12 . 你有用过哪些前端性能优化的方法？**

（1） 减少http请求次数：CSS Sprites, JS、CSS源码压缩、图片大小控制合适；网页Gzip，CDN托管，data缓存 ，图片服务器。   
（2） 前端模板 JS+数据，减少由于HTML标签导致的带宽浪费，前端用变量保存AJAX请求结果，每次操作本地变量，不用请求，减少请求次数   
（3） 用innerHTML代替DOM操作，减少DOM操作次数，优化javascript性能。   
（4） 当需要设置的样式很多时设置className而不是直接操作style。   
（5） 少用全局变量、缓存DOM节点查找的结果。减少IO读取操作。   
（6） 避免使用CSS Expression（css表达式)又称Dynamic properties(动态属性)。   
（7） 图片预加载，将样式表放在顶部，将脚本放在底部 加上时间戳。   
（8） 避免在页面的主体布局中使用table，table要等其中的内容完全下载之后才会显示出来，显示比div+css布局慢。

**14 . 一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么？（流程说的越详细越好）**

查找浏览器缓存   
 DNS解析、查找该域名对应的IP地址、重定向（301）、发出第二个GET请求   
 进行HTTP协议会话   
 客户端发送报头(请求报头)   
 服务器回馈报头(响应报头)   
 html文档开始下载   
 文档树建立，根据标记请求所需指定MIME类型的文件   
 文件显示   
浏览器这边做的工作大致分为以下几步：   
 加载：根据请求的URL进行域名解析，向服务器发起请求，接收文件（HTML、JS、CSS、图象等）。   
 解析：对加载到的资源（HTML、JS、CSS等）进行语法解析，建议相应的内部**[数据结构](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)**（比如HTML的DOM树，JS的（对象）属性表，CSS的样式规则等等） 

**23 . 平时如何管理你的项目？** 先期团队必须确定好全局样式（globe.css），编码模式(utf-8) 等；   
 编写习惯必须一致（例如都是采用继承式的写法，单样式都写成一行）；   
 标注样式编写人，各模块都及时标注（标注关键样式调用的地方）；   
 页面进行标注（例如 页面 模块 开始和结束）；   
 CSS跟HTML 分文件夹并行存放，命名都得统一（例如style.css）；   
 JS 分文件夹存放 命名以该JS功能为准的英文翻译。   
 图片采用整合的 images.png png8 格式文件使用 尽量整合在一起使用方便将来的管理

**24 . 说说最近最流行的一些东西吧？常去哪些网站？** Node.js、**[MongoDB](http://lib.csdn.net/base/mongodb" \o "MongoDB知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)**、npm、MVVM、MEAN、three.js   
25**.移动端（Android IOS）怎么做好用户体验?**

清晰的视觉纵线、信息的分组、极致的减法、   
 利用选择代替输入、标签及文字的排布方式、   
 依靠明文确认密码、合理的键盘利用

**26、想问公司的问题？** 目前关注哪些最新的Web前端技术（未来的发展方向）？   
 前端团队如何工作的（实现一个产品的流程）？   
 公司的薪资结构是什么样子的？

## HTML/CSS部分

**1、什么是盒子模型？**

在网页中，一个元素占有空间的大小由几个部分构成，其中包括元素的内容（content），元素的内边距（padding），元素的边框（border），元素的外边距（margin）四个部分。这四个部分占有的空间中，有的部分可以显示相应的内容，而有的部分只用来分隔相邻的区域或区域。4个部分一起构成了css中元素的盒模型。

**2、行内元素有哪些？块级元素有哪些？ 空(void)元素有那些？**

行内元素：a、b、span、img、input、strong、select、label、em、button、textarea   
 块级元素：div、ul、li、dl、dt、dd、p、h1-h6、blockquote   
 空元素：即系没有内容的HTML元素，例如：br、meta、hr、link、input、img

**3、CSS实现垂直水平居中**

一道经典的问题，实现方法有很多种，以下是其中一种实现：   
 HTML结构：

<div class="wrapper">

<div class="content"></div>

</div>

CSS：

.wrapper{position:relative;}

.content{ background-color:#6699FF;

width:200px;

height:200px;

position: absolute; //父元素需要相对定位

top: 50%;

left: 50%;

margin-top:-100px ; //二分之一的height，width

margin-left: -100px;

}

**4、简述一下src与href的区别**

href 是指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，用于超链接。

src是指向外部资源的位置，指向的内容将会嵌入到文档中当前标签所在位置；在请求src资源时会将其指向的资源下载并应用到文档内，例如js脚本，img图片和frame等元素。当浏览器解析到该元素时，会暂停其他资源的下载和处理，直到将该资源加载、编译、执行完毕，图片和框架等元素也如此，类似于将所指向资源嵌入当前标签内。这也是为什么将js脚本放在底部而不是头部。

**5、什么是CSS Hack?**

一般来说是针对不同的浏览器写不同的CSS,就是 CSS Hack。   
IE浏览器Hack一般又分为三种，条件Hack、属性级Hack、选择符Hack（详细参考CSS文档：css文档）。

例如：

1、条件Hack

<!--[if IE]>

<style>

.test{color:red;}

</style>

<![endif]>

2、属性Hack

.test{

color:#090\9; / For IE8+ /

color:#f00; / For IE7 and earlier /

\_color:#ff0; / For IE6 and earlier \*/

}

3、选择符Hack

html .test{color:#090;} / For IE6 and earlier /

html .test{color:#ff0;} / For IE7 /

**6、简述同步和异步的区别**

同步是阻塞模式，异步是非阻塞模式。   
 同步就是指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么这个进程将会一直等待下去，直到收到返回信息才继续执行下去；   
 异步是指进程不需要一直等下去，而是继续执行下面的操作，不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执行的效率。

**7、px和em的区别**

px和em都是长度单位，区别是，px的值是固定的，指定是多少就是多少，计算比较容易。em得值不是固定的，并且em会继承父级元素的字体大小。 浏览器的默认字体高都是16px。所以未经调整的浏览器都符合: 1em=16px。那么12px=0.75em, 10px=0.625em

**8、什么叫优雅降级和渐进增强？**

渐进增强 progressive enhancement：

针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追 加功能达到更好的用户体验。

优雅降级 graceful degradation：

一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。

区别：   
 a. 优雅降级是从复杂的现状开始，并试图减少用户体验的供给   
 b. 渐进增强则是从一个非常基础的，能够起作用的版本开始，并不断扩充，以适应未来环境的需要   
 c. 降级（功能衰减）意味着往回看；而渐进增强则意味着朝前看，同时保证其根基处于安全地带

**9、浏览器的内核分别是什么?**

IE: trident内核   
 Firefox：gecko内核   
 Safari：webkit内核   
 Opera：以前是presto内核，Opera现已改用Google Chrome的Blink内核   
 Chrome：Blink(基于webkit，Google与Opera Software共同开发)

## ****JavaScript部分****

**1、怎样添加、移除、移动、复制、创建和查找节点？**

1）创建新节点

createDocumentFragment()//创建一个DOM片段

createElement() //创建一个具体的元素

createTextNode() //创建一个文本节点

2）添加、移除、替换、插入

appendChild() //添加

removeChild() //移除

replaceChild() //替换

insertBefore() //插入

3）查找

getElementsByTagName() //通过标签名称

getElementsByName() //通过元素的Name属性的值

getElementById() //通过元素Id，唯一性

**2、实现一个函数clone，可以对[JavaScript](http://lib.csdn.net/base/javascript" \o "JavaScript知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)中的5种主要的数据类型（包括Number、String、Object、Array、Boolean）进行值复制。**

/\*\*

\* 对象克隆

\* 支持基本数据类型及对象

\* 递归方法

\*/

function clone(obj) {

var o;

switch (typeof obj) {

case "undefined":

break;

case "string":

o = obj + "";

break;

case "number":

o = obj - 0;

break;

case "boolean":

o = obj;

break;

case "object": // object 分为两种情况 对象（Object）或数组（Array）

if (obj === null) {o = null; }

else {

if(Object.prototype.toString.call(obj).slice(8, -1) === "Array") {

o = [];

for (var i = 0; i < obj.length; i++){o.push(clone(obj[i]));}

}

else {

o = {};

for (var k in obj) {o[k] = clone(obj[k]);}

}

}

break;

default:

o = obj;

break;

} //switch函数结束

return o;

}

**3、如何消除一个数组里面重复的元素？**

var arr1 =[1,2,2,2,3,3,3,4,5,6],

arr2 = []; for(var i = 0,len = arr1.length; i< len; i++){

if(arr2.indexOf(arr1[i]) < 0){

arr2.push(arr1[i]);

}

}

document.write(arr2); // 1,2,3,4,5,6

**5、在Javascript中什么是伪数组？如何将伪数组转化为标准数组？**

伪数组（类数组）：无法直接调用数组方法或期望length属性有什么特殊的行为，但仍可以对真正数组遍历方法来遍历它们。典型的是函数的argument参数，还有像调用getElementsByTagName,document.childNodes之类的,它们都返回NodeList对象都属于伪数组。可以使用Array.prototype.slice.call(fakeArray)将数组转化为真正的Array对象。

function log(){ //为了使用unshift数组方法，将argument转化为真正的数组

var args = Array.prototype.slice.call(arguments);

args.unshift('(app)');

console.log.apply(console, args);

};

**6、Javascript中callee和caller的作用？**

caller是返回一个对函数的引用，该函数调用了当前函数；   
 callee是返回正在被执行的function函数，也就是所指定的function对象的正文。

**7、请描述一下cookies，sessionStorage和localStorage的区别**

sessionStorage用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此sessionStorage不是一种持久化的本地存储，仅仅是会话级别的存储。 localStorage用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。 web storage和cookie的区别 Web Storage的概念和cookie相似，区别是它是为了更大容量存储设计的。Cookie的大小是受限的，并且每次你请求一个新的页面的时候Cookie都会被发送过去，这样无形中浪费了带宽，另外cookie还需要指定作用域，不可以跨域调用。 除此之外，Web Storage拥有setItem,getItem,removeItem,clear等方法，不像cookie需要[前端开发](http://lib.csdn.net/base/jquery" \o "jQuery知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)者自己封装setCookie，getCookie。但是Cookie也是不可以或缺的：Cookie的作用是与服务器进行交互，作为HTTP规范的一部分而存在 ，而Web Storage仅仅是为了在本地“存储”数据而生。

1. **手写数组快速排序**  
    “快速排序”的思想很简单，整个排序过程只需要三步：   
    （1）在数据集之中，选择一个元素作为”基准”（pivot）。   
    （2）所有小于”基准”的元素，都移到”基准”的左边；所有大于”基准”的元素，都移到”基准”的右边。   
    （3）对”基准”左边和右边的两个子集，不断重复第一步和第二步，直到所有子集只剩下一个元素为止。

**9、统计字符串”aaaabbbccccddfgh”中字母个数或统计最多字母数。**

var str = "aaaabbbccccddfgh";

var obj = {};

for(var i=0;i<str.length;i++){

var v = str.charAt(i);

if(obj[v]&&obj[v].value==v){

obj[v].count=++obj[v].count;

}else{

obj[v]={};

obj[v].count=1;

obj[v].value=v;

}

}

for(key in obj){

ocumemt.write(obj[key].value+'='+obj[key].count+'');//a=4 b=3 c=4 d=2 f=1 g=1 h=1

}

**10、写一个function，清除字符串前后的空格。（兼容所有浏览器）**

function trim(str){

if(str&&typeof str ===="string"){

return str.replace(/(^\s)/(ls)$/g,"");//去除前后空白符

}

}

**1、一次完整的HTTP事务是怎样的一个过程？**基本流程：   
 a. 域名解析   
 b. 发起TCP的3次握手   
 c. 建立TCP连接后发起http请求   
 d. 服务器端响应http请求，浏览器得到html代码   
 e. 浏览器解析html代码，并请求html代码中的资源   
 f. 浏览器对页面进行渲染呈现给用户

**2、对前端工程师这个职位你是怎么样理解的？** a. 前端是最贴近用户的程序员，前端的能力就是能让产品从 90分进化到 100 分，甚至更好   
 b. 参与项目，快速高质量完成实现效果图，精确到1px；   
 c. 与团队成员，UI设计，产品经理的沟通；   
 d. 做好的页面结构，页面重构和用户体验；   
 e. 处理hack，兼容、写出优美的代码格式；   
 f. 针对服务器的优化、拥抱最新前端技术。

**19. 为什么要使用Div+CSS布局** 形式与内容分离  
 大大减少页面代码，提高页面浏览速度  
 结构清晰，有利于SEO  
 缩短改版时间， 布局更方便  
 一次设计，多次使用

**20. Block元素的特点是什么?哪些元素默认为Block元素**总是在新行上开始；  
高度，行高以及顶和底边距都可控制；  
宽度缺省是它的容器的100%，除非设定一个宽度  
是块元素的有：,,

inline元素的特点是什么?哪些元素属于inline元素?  
和其他元素都在一行上；  
高，行高及顶和底边距不可改变；  
宽度就是它的文字或图片的宽度，不可改变。

支持h5、c3的浏览器

chrome、Firefox、360、opera

会话跟踪常用的4种方法：

URL重写：URL重写技术就是在URL结尾添加一个附加数据以标识该会话，把会话ID通过URL的信息传递过去，以便在服务端进行识别不同的用户。

隐藏表单域：将会话ID添加到HTML表单元素中提交到服务器，此表单不再客户端显示。

Cookie：Cookie是Web服务器发送给客户端的一小段信息，客户端请求时可以读取该信息发送到服务器端，进而进行用户的识别。对于客户端的每次请求，服务器都会将Cookie发送到客户端，在客户端可以进行保存,以便下次使用。

session: 在服务器端会创建一个session对象，产生一个sessionID来标识这个session对象，然后将这个sessionID放入到Cookie中发送到客户端，下一次访问时，sessionID会发送到服务器，在服务器端进行识别不同的用户 , Session是依赖Cookie的，如果Cookie被禁用，那么session也将失效 】

1.自我介绍：除了基本个人信息以外，面试官更想听的是你与众不同的地方和你的优势。

2.项目介绍

3.如何看待前端开发？

4.平时是如何学习前端开发的？

5.未来三到五年的规划是怎样的？

**1、如何解决跨域问题**

①JSONP：

原理是：动态插入script标签，通过script标签引入一个js文件，这个js文件载入成功后会执行我们在url参数中指定的函数，并且会把我们需要的json数据作为参数传入。

由于同源策略的限制，XmlHttpRequest只允许请求当前源（域名、协议、端口）的资源，为了实现跨域请求，可以通过script标签实现跨域请求，然后在服务端输出JSON数据并执行回调函数，从而解决了跨域的数据请求。

优点是兼容性好，简单易用，支持浏览器与服务器双向通信。缺点是只支持GET请求。

JSONP：json+padding（内填充），顾名思义，就是把JSON填充到一个盒子里

②CORS

服务器端对于CORS的支持，主要就是通过设置Access-Control-Allow-Origin来进行的。如果浏览器检测到相应的设置，就可以允许Ajax进行跨域的访问。

③通过修改document.domain来跨子域

将子域和主域的document.domain设为同一个主域.前提条件：这两个域名必须属于同一个基础域名!而且所用的协议，端口都要一致，否则无法利用document.domain进行跨域

主域相同的使用document.domain

④使用window.name来进行跨域

window对象有个name属性，该属性有个特征：即在一个窗口(window)的生命周期内,窗口载入的所有的页面都是共享一个window.name的，每个页面对window.name都有读写的权限，window.name是持久存在一个窗口载入过的所有页面中的

⑤使用HTML5中新引进的window.postMessage方法来跨域传送数据

还有flash、在服务器上设置代理页面等跨域方式。个人认为window.name的方法既不复杂，也能兼容到几乎所有浏览器，这真是极好的一种跨域方法。

**2、XML和JSON的区别？**

(1).数据体积方面。

JSON相对于XML来讲，数据的体积小，传递的速度更快些。

(2).数据交互方面。

JSON与JavaScript的交互更加方便，更容易解析处理，更好的数据交互。

(3).数据描述方面。

JSON对数据的描述性比XML较差。

(4).传输速度方面。

JSON的速度要远远快于XML。

**3、谈谈你对webpack的看法**

WebPack 是一个模块打包工具，你可以使用WebPack管理你的模块依赖，并编绎输出模块们所需的静态文件。它能够很好地管理、打包Web开发中所用到的HTML、Javascript、CSS以及各种静态文件（图片、字体等），让开发过程更加高效。对于不同类型的资源，webpack有对应的模块加载器。webpack模块打包器会分析模块间的依赖关系，最后 生成了优化且合并后的静态资源。

webpack的两大特色：

1.code splitting（可以自动完成）

2.loader 可以处理各种类型的静态文件，并且支持串联操作

webpack 是以commonJS的形式来书写脚本滴，但对 AMD/CMD 的支持也很全面，方便旧项目进行代码迁移。

webpack具有requireJs和browserify的功能，但仍有很多自己的新特性：

1. 对 CommonJS 、 AMD 、ES6的语法做了兼容

2. 对js、css、图片等资源文件都支持打包

3. 串联式模块加载器以及插件机制，让其具有更好的灵活性、扩展性，例如对CoffeeScript、ES6的支持

4. 有独立的配置文件webpack.config.js

5. 可以将代码切割成不同的chunk，实现按需加载，降低了初始化时间

6. 支持 SourceUrls 和 SourceMaps，易于调试

7. 具有强大的Plugin接口，大多是内部插件，使用起来比较灵活

8.webpack 使用异步 IO 并具有多级缓存。这使得 webpack 很快且在增量编译上更加快

**4、说说TCP传输的三次握手四次挥手策略**

为了准确无误地把数据送达目标处，TCP协议采用了三次握手策略。用TCP协议把数据包送出去后，TCP不会对传送 后的情况置之不理，它一定会向对方确认是否成功送达。握手过程中使用了TCP的标志：SYN和ACK。

发送端首先发送一个带SYN标志的数据包给对方。接收端收到后，回传一个带有SYN/ACK标志的数据包以示传达确认信息。   
最后，发送端再回传一个带ACK标志的数据包，代表“握手”结束。   
若在握手过程中某个阶段莫名中断，TCP协议会再次以相同的顺序发送相同的数据包。

断开一个TCP连接则需要“四次握手”：

第一次挥手：主动关闭方发送一个FIN，用来关闭主动方到被动关闭方的数据传送，也就是主动关闭方告诉被动关闭方：我已经不 会再给你发数据了(当然，在fin包之前发送出去的数据，如果没有收到对应的ack确认报文，主动关闭方依然会重发这些数据)，但是，此时主动关闭方还可 以接受数据。

第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号）。

第三次挥手：被动关闭方发送一个FIN，用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会再给你发数据了。

第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手。

TCP和UDP的区别

TCP（Transmission Control Protocol，传输控制协议）是基于连接的协议，也就是说，在正式收发数据前，必须和对方建立可靠的连接。一个TCP连接必须要经过三次“对话”才能建立起来

UDP（User Data Protocol，用户数据报协议）是与TCP相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去！   
UDP适用于一次只传送少量数据、对可靠性要求不高的应用环境。

**5、说说你对作用域链的理解**

作用域链的作用是保证执行环境里有权访问的变量和函数是有序的，作用域链的变量只能向上访问，变量访问到window对象即被终止，作用域链向下访问变量是不被允许的。

**6、创建ajax过程**

(1)创建XMLHttpRequest对象,也就是创建一个异步调用对象.

(2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息.

(3)设置响应HTTP请求状态变化的函数.

(4)发送HTTP请求.

(5)获取异步调用返回的数据.

(6)使用JavaScript和DOM实现局部刷新.

**7、渐进增强和优雅降级**

渐进增强 ：针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。

优雅降级 ：一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。

**常见web安全及防护原理**

**1、sql注入原理**

就是通过把SQL命令插入到Web表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。

总的来说有以下几点：

1.永远不要信任用户的输入，要对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双"-"进行转换等。

2.永远不要使用动态拼装SQL，可以使用参数化的SQL或者直接使用存储过程进行数据查询存取。

3.永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。

4.不要把机密信息明文存放，请加密或者hash掉密码和敏感的信息。

**2、XSS原理及防范**

Xss(cross-site scripting)攻击指的是攻击者往Web页面里插入恶意 html标签或者javascript代码。

比如：攻击者在论坛中放一个看似安全的链接，骗取用户点击后，窃取cookie中的用户私密信息；或者攻击者在论坛中加一个恶意表单，当用户提交表单的时候，却把信息传送到攻击者的服务器中，而不是用户原本以为的信任站点。

**3、XSS防范方法**

首先代码里对用户输入的地方和变量都需要仔细检查长度和对”<”,”>”,”;”,”’”等字符做过滤；其次任何内容写到页面之前都必须加以encode，避免不小心把html tag 弄出来。这一个层面做好，至少可以堵住超过一半的XSS 攻击。

其次，避免直接在cookie 中泄露用户隐私，例如email、密码等等。

再次，通过使cookie 和系统ip 绑定来降低cookie 泄露后的危险。这样攻击者得到的cookie 没有实际价值，不可能拿来重放。

如果网站不需要再浏览器端对cookie 进行操作，可以在Set-Cookie 末尾加上HttpOnly 来防止javascript 代码直接获取cookie 。

尽量采用POST 而非GET 提交表单

**4、XSS与CSRF有什么区别吗？**

XSS是获取信息，不需要提前知道其他用户页面的代码和数据包。CSRF是代替用户完成指定的动作，需要知道其他用户页面的代码和数据包。

要完成一次CSRF攻击，受害者必须依次完成两个步骤：

登录受信任网站A，并在本地生成Cookie。

在不登出A的情况下，访问危险网站B。

**5、CSRF的防御**

服务端的CSRF方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数。

通过验证码的方法

**6、Web Worker 和webSocket**

worker主线程:

1.通过 worker = new Worker( url ) 加载一个JS文件来创建一个worker，同时返回一个worker实例。

2.通过worker.postMessage( data ) 方法来向worker发送数据

3.绑定worker.onmessage方法来接收worker发送过来的数据。

4.可以使用 worker.terminate() 来终止一个worker的执行。

WebSocket是Web应用程序的传输协议，它提供了双向的，按序到达的数据流。他是一个HTML5协议，WebSocket的连接是持久的，他通过在客户端和服务器之间保持双工连接，服务器的更新可以被及时推送给客户端，而不需要客户端以一定时间间隔去轮询。

**7、HTTP和HTTPS**

HTTP协议通常承载于TCP协议之上，在HTTP和TCP之间添加一个安全协议层（SSL或TSL），这个时候，就成了我们常说的HTTPS。

默认HTTP的端口号为80，HTTPS的端口号为443。

为什么HTTPS安全

因为网络请求需要中间有很多的服务器路由器的转发。中间的节点都可能篡改信息，而如果使用HTTPS，密钥在你和终点站才有。https之所以比http安全，是因为他利用ssl/tls协议传输。它包含证书，卸载，流量转发，负载均衡，页面适配，浏览器适配，refer传递等。保障了传输过程的安全性

**8、对前端模块化的认识**

AMD 是 RequireJS 在推广过程中对模块定义的规范化产出。

CMD 是 SeaJS 在推广过程中对模块定义的规范化产出。

AMD 是提前执行，CMD 是延迟执行。

AMD推荐的风格通过返回一个对象做为模块对象，CommonJS的风格通过对module.exports或exports的属性赋值来达到暴露模块对象的目的。

CMD模块方式

define(function(require, exports, module) {

// 模块代码

});

**9、Javascript垃圾回收方法**

标记清除（mark and sweep）

这是JavaScript最常见的垃圾回收方式，当变量进入执行环境的时候，比如函数中声明一个变量，垃圾回收器将其标记为“进入环境”，当变量离开环境的时候（函数执行结束）将其标记为“离开环境”。

垃圾回收器会在运行的时候给存储在内存中的所有变量加上标记，然后去掉环境中的变量以及被环境中变量所引用的变量（闭包），在这些完成之后仍存在标记的就是要删除的变量了

引用计数(reference counting)

在低版本IE中经常会出现内存泄露，很多时候就是因为其采用引用计数方式进行垃圾回收。引用计数的策略是跟踪记录每个值被使用的次数，当声明了一个 变量并将一个引用类型赋值给该变量的时候这个值的引用次数就加1，如果该变量的值变成了另外一个，则这个值得引用次数减1，当这个值的引用次数变为0的时 候，说明没有变量在使用，这个值没法被访问了，因此可以将其占用的空间回收，这样垃圾回收器会在运行的时候清理掉引用次数为0的值占用的空间。

在IE中虽然JavaScript对象通过标记清除的方式进行垃圾回收，但BOM与DOM对象却是通过引用计数回收垃圾的，   
也就是说只要涉及BOM及DOM就会出现循环引用问题。

**10、你觉得前端工程的价值体现在哪**

为简化用户使用提供技术支持（交互部分）

为多个浏览器兼容性提供支持

为提高用户浏览速度（浏览器性能）提供支持

为跨平台或者其他基于webkit或其他渲染引擎的应用提供支持

为展示数据提供支持（数据接口）

**11、谈谈性能优化问题**

代码层面：避免使用css表达式，避免使用高级选择器，通配选择器。

缓存利用：缓存Ajax，使用CDN，使用外部js和css文件以便缓存，添加Expires头，服务端配置Etag，减少DNS查找等

请求数量：合并样式和脚本，使用css图片精灵，初始首屏之外的图片资源按需加载，静态资源延迟加载。

请求带宽：压缩文件，开启GZIP，

A、代码层面的优化

用hash-table来优化查找

少用全局变量

用innerHTML代替DOM操作，减少DOM操作次数，优化javascript性能

用setTimeout来避免页面失去响应

缓存DOM节点查找的结果

避免使用CSS Expression

避免全局查询

避免使用with(with会创建自己的作用域，会增加作用域链长度)

多个变量声明合并

避免图片和iFrame等的空Src。空Src会重新加载当前页面，影响速度和效率

尽量避免写在HTML标签中写Style属性

移动端性能优化

尽量使用css3动画，开启硬件加速。

适当使用touch事件代替click事件。

避免使用css3渐变阴影效果。

可以用transform: translateZ(0)来开启硬件加速。

不滥用Float。Float在渲染时计算量比较大，尽量减少使用

不滥用Web字体。Web字体需要下载，解析，重绘当前页面，尽量减少使用。

合理使用requestAnimationFrame动画代替setTimeout

CSS中的属性（CSS3 transitions、CSS3 3D transforms、Opacity、Canvas、WebGL、Video）会触发GPU渲染，请合理使用。过渡使用会引发手机过耗电增加

PC端的在移动端同样适用

**12、什么是Etag？**

当发送一个服务器请求时，浏览器首先会进行缓存过期判断。浏览器根据缓存过期时间判断缓存文件是否过期。

情景一：若没有过期，则不向服务器发送请求，直接使用缓存中的结果，此时我们在浏览器控制台中可以看到 200 OK(from cache) ，此时的情况就是完全使用缓存，浏览器和服务器没有任何交互的。

情景二：若已过期，则向服务器发送请求，此时请求中会带上①中设置的文件修改时间，和Etag

然后，进行资源更新判断。服务器根据浏览器传过来的文件修改时间，判断自浏览器上一次请求之后，文件是不是没有被修改过；根据Etag，判断文件内容自上一次请求之后，有没有发生变化

情形一：若两种判断的结论都是文件没有被修改过，则服务器就不给浏览器发index.html的内容了，直接告诉它，文件没有被修改过，你用你那边的缓存吧—— 304 Not Modified，此时浏览器就会从本地缓存中获取index.html的内容。此时的情况叫协议缓存，浏览器和服务器之间有一次请求交互。

情形二：若修改时间和文件内容判断有任意一个没有通过，则服务器会受理此次请求，之后的操作同①

① 只有get请求会被缓存，post请求不会

Expires和Cache-Control

Expires要求客户端和服务端的时钟严格同步。HTTP1.1引入Cache-Control来克服Expires头的限制。如果max-age和Expires同时出现，则max-age有更高的优先级。

Cache-Control: no-cache, private, max-age=0

ETag: abcde

Expires: Thu, 15 Apr 2014 20:00:00 GMT

Pragma: private

Last-Modified: $now // RFC1123 format

ETag应用:

Etag由服务器端生成，客户端通过If-Match或者说If-None-Match这个条件判断请求来验证资源是否修改。常见的是使用If-None-Match。请求一个文件的流程可能如下：

====第一次请求===

1.客户端发起 HTTP GET 请求一个文件；

2.服务器处理请求，返回文件内容和一堆Header，当然包括Etag(例如"2e681a-6-5d044840")(假设服务器支持Etag生成和已经开启了Etag).状态码200

====第二次请求===

客户端发起 HTTP GET 请求一个文件，注意这个时候客户端同时发送一个If-None-Match头，这个头的内容就是第一次请求时服务器返回的Etag：2e681a-6-5d0448402.服务器判断发送过来的Etag和计算出来的Etag匹配，因此If-None-Match为False，不返回200，返回304，客户端继续使用本地缓存；流程很简单，问题是，如果服务器又设置了Cache-Control:max-age和Expires呢，怎么办

答案是同时使用，也就是说在完全匹配If-Modified-Since和If-None-Match即检查完修改时间和Etag之后，

服务器才能返回304.(不要陷入到底使用谁的问题怪圈)

为什么使用Etag请求头?

Etag 主要为了解决 Last-Modified 无法解决的一些问题。

**13、栈和队列的区别?**

栈的插入和删除操作都是在一端进行的，而队列的操作却是在两端进行的。

队列先进先出，栈先进后出。

栈只允许在表尾一端进行插入和删除，而队列只允许在表尾一端进行插入，在表头一端进行删除

**14、栈和堆的区别？**

栈区（stack）— 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。

堆区（heap） — 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收。

堆（数据结构）：堆可以被看成是一棵树，如：堆排序；

栈（数据结构）：一种先进后出的数据结构。

快速 排序的思想并实现一个快排？

“快速排序”的思想很简单，整个排序过程只需要三步：

　　（1）在数据集之中，找一个基准点

　　（2）建立两个数组，分别存储左边和右边的数组

　　（3）利用递归进行下次比较

<script type="text/javascript">

function quickSort(arr){

if(arr.length<=1){

return arr;//如果数组只有一个数，就直接返回；

}

var num = Math.floor(arr.length/2);//找到中间数的索引值，如果是浮点数，则向下取整

var numValue = arr.splice(num,1);//找到中间数的值

var left = [];

var right = [];

for(var i=0;i<arr.length;i++){

if(arr[i]<numValue){

left.push(arr[i]);//基准点的左边的数传到左边数组

}

else{

right.push(arr[i]);//基准点的右边的数传到右边数组

}

}

return quickSort(left).concat([numValue],quickSort(right));//递归不断重复比较

}

alert(quickSort([32,45,37,16,2,87]));//弹出“2,16,32,37,45,87”

</script>

**15、你觉得jQuery或zepto源码有哪些写的好的地方**

jquery源码封装在一个匿名函数的自执行环境中，有助于防止变量的全局污染，然后通过传入window对象参数，可以使window对象作为局部变量使用，好处是当jquery中访问window对象的时候，就不用将作用域链退回到顶层作用域了，从而可以更快的访问window对象。同样，传入undefined参数，可以缩短查找undefined时的作用域链。

(function( window, undefined ) {

//用一个函数域包起来，就是所谓的沙箱

//在这里边var定义的变量，属于这个函数域内的局部变量，避免污染全局

//把当前沙箱需要的外部变量通过函数参数引入进来

//只要保证参数对内提供的接口的一致性，你还可以随意替换传进来的这个参数

window.jQuery = window.$ = jQuery;

})( window );

jquery将一些原型属性和方法封装在了jquery.prototype中，为了缩短名称，又赋值给了jquery.fn，这是很形象的写法。

有一些数组或对象的方法经常能使用到，jQuery将其保存为局部变量以提高访问速度。

jquery实现的链式调用可以节约代码，所返回的都是同一个对象，可以提高代码效率。

**16、ES6的了解**

新增模板字符串（为JavaScript提供了简单的字符串插值功能）、箭头函数（操作符左边为输入的参数，而右边则是进行的操作以及返回的值Inputs=>outputs。）、for-of（用来遍历数据—例如数组中的值。）arguments对象可被不定参数和默认参数完美代替。ES6将promise对象纳入规范，提供了原生的Promise对象。增加了let和const命令，用来声明变量。增加了块级作用域。let命令实际上就增加了块级作用域。ES6规定，var命令和function命令声明的全局变量，属于全局对象的属性；let命令、const命令、class命令声明的全局变量，不属于全局对象的属性。。还有就是引入module模块的概念

**17、js继承方式及其优缺点**

原型链继承的缺点

一是字面量重写原型会中断关系，使用引用类型的原型，并且子类型还无法给超类型传递参数。

借用构造函数（类式继承）

借用构造函数虽然解决了刚才两种问题，但没有原型，则复用无从谈起。所以我们需要原型链+借用构造函数的模式，这种模式称为组合继承

组合式继承

组合式继承是比较常用的一种继承方法，其背后的思路是 使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承。这样，既通过在原型上定义方法实现了函数复用，又保证每个实例都有它自己的属性。

**18、关于Http 2.0 你知道多少？**

HTTP/2引入了“服务端推（server push）”的概念，它允许服务端在客户端需要数据之前就主动地将数据发送到客户端缓存中，从而提高性能。

HTTP/2提供更多的加密支持

HTTP/2使用多路技术，允许多个消息在一个连接上同时交差。

它增加了头压缩（header compression），因此即使非常小的请求，其请求和响应的header都只会占用很小比例的带宽。

**19、defer和async**

defer并行加载js文件，会按照页面上script标签的顺序执行   
async并行加载js文件，下载完成立即执行，不会按照页面上script标签的顺序执行

**20、谈谈浮动和清除浮动**

浮动的框可以向左或向右移动，直到他的外边缘碰到包含框或另一个浮动框的边框为止。由于浮动框不在文档的普通流中，所以文档的普通流的块框表现得就像浮动框不存在一样。浮动的块框会漂浮在文档普通流的块框上。

**21、如何评价AngularJS和BackboneJS**

backbone具有依赖性，依赖underscore.js。Backbone + Underscore + jQuery(or Zepto) 就比一个AngularJS 多出了2 次HTTP请求.

Backbone的Model没有与UI视图数据绑定，而是需要在View中自行操作DOM来更新或读取UI数据。AngularJS与此相反，Model直接与UI视图绑定，Model与UI视图的关系，通过directive封装，AngularJS内置的通用directive，就能实现大部分操作了，也就是说，基本不必关心Model与UI视图的关系，直接操作Model就行了，UI视图自动更新。

AngularJS的directive，你输入特定数据，他就能输出相应UI视图。是一个比较完善的前端MVW框架，包含模板，数据双向绑定，路由，模块化，服务，依赖注入等所有功能，模板功能强大丰富，并且是声明式的，自带了丰富的 Angular 指令。

**22、用过哪些设计模式？**

工厂模式：

主要好处就是可以消除对象间的耦合，通过使用工程方法而不是new关键字。将所有实例化的代码集中在一个位置防止代码重复。

工厂模式解决了重复实例化的问题 ，但还有一个问题,那就是识别问题，因为根本无法 搞清楚他们到底是哪个对象的实例。

function createObject(name,age,profession){//集中实例化的函数var obj = new Object();

obj.name = name;

obj.age = age;

obj.profession = profession;

obj.move = function () {

return this.name + ' at ' + this.age + ' engaged in ' + this.profession;

};

return obj;

}

var test1 = createObject('trigkit4',22,'programmer');//第一个实例var test2 = createObject('mike',25,'engineer');//第二个实例

构造函数模式

使用构造函数的方法 ，即解决了重复实例化的问题 ，又解决了对象识别的问题，该模式与工厂模式的不同之处在于：

1.构造函数方法没有显示的创建对象 (new Object());

2.直接将属性和方法赋值给 this 对象;

3.没有 renturn 语句。

**23、说说你对闭包的理解**

使用闭包主要是为了设计私有的方法和变量。闭包的优点是可以避免全局变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在js中，函数即闭包，只有函数才会产生作用域的概念

闭包有三个特性：

1.函数嵌套函数

2.函数内部可以引用外部的参数和变量

3.参数和变量不会被F垃圾回收机制回收

**24、请你谈谈Cookie的弊端**

cookie虽然在持久保存客户端数据提供了方便，分担了服务器存储的负担，但还是有很多局限性的。

第一：每个特定的域名下最多生成20个cookie

1.IE6或更低版本最多20个cookie

2.IE7和之后的版本最后可以有50个cookie。

3.Firefox最多50个cookie

4.chrome和Safari没有做硬性限制

IE和Opera 会清理近期最少使用的cookie，Firefox会随机清理cookie。

cookie的最大大约为4096字节，为了兼容性，一般不能超过4095字节。

IE 提供了一种存储可以持久化用户数据，叫做userdata，从IE5.0就开始支持。每个数据最多128K，每个域名下最多1M。这个持久化数据放在缓存中，如果缓存没有清理，那么会一直存在。

优点：极高的扩展性和可用性

1.通过良好的编程，控制保存在cookie中的session对象的大小。

2.通过加密和安全传输技术（SSL），减少cookie被破解的可能性。

3.只在cookie中存放不敏感数据，即使被盗也不会有重大损失。

4.控制cookie的生命期，使之不会永远有效。偷盗者很可能拿到一个过期的cookie。

缺点：

1.`Cookie`数量和长度的限制。每个domain最多只能有20条cookie，每个cookie长度不能超过4KB，否则会被截掉.

2.安全性问题。如果cookie被人拦截了，那人就可以取得所有的session信息。即使加密也与事无补，因为拦截者并不需要知道cookie的意义，他只要原样转发cookie就可以达到目的了。

3.有些状态不可能保存在客户端。例如，为了防止重复提交表单，我们需要在服务器端保存一个计数器。如果我们把这个计数器保存在客户端，那么它起不到任何作用。

**25、浏览器本地存储**

在较高版本的浏览器中，js提供了sessionStorage和globalStorage。在HTML5中提供了localStorage来取代globalStorage。

html5中的Web Storage包括了两种存储方式：sessionStorage和localStorage。

sessionStorage用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此sessionStorage不是一种持久化的本地存储，仅仅是会话级别的存储。

而localStorage用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。

web storage和cookie的区别

Web Storage的概念和cookie相似，区别是它是为了更大容量存储设计的。Cookie的大小是受限的，并且每次你请求一个新的页面的时候Cookie都会被发送过去，这样无形中浪费了带宽，另外cookie还需要指定作用域，不可以跨域调用。

除此之外，Web Storage拥有setItem,getItem,removeItem,clear等方法，不像cookie需要前端开发者自己封装setCookie，getCookie。

但是cookie也是不可以或缺的：cookie的作用是与服务器进行交互，作为HTTP规范的一部分而存在 ，而Web Storage仅仅是为了在本地“存储”数据而生

浏览器的支持除了IE７及以下不支持外，其他标准浏览器都完全支持(ie及FF需在web服务器里运行)，值得一提的是IE总是办好事，例如IE7、IE6中的userData其实就是javascript本地存储的解决方案。通过简单的代码封装可以统一到所有的浏览器都支持web storage。

localStorage和sessionStorage都具有相同的操作方法，例如setItem、getItem和removeItem等

cookie 和session 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗

考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能

考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、所以个人建议：

将登陆信息等重要信息存放为SESSION

其他信息如果需要保留，可以放在COOKIE中

**26、display:none和visibility:hidden的区别？**

display:none 隐藏对应的元素，在文档布局中不再给它分配空间，它各边的元素会合拢，就当他从来不存在。

visibility:hidden 隐藏对应的元素，但是在文档布局中仍保留原来的空间。

**27、CSS中link 和@import的区别是？**

(1) link属于HTML标签，而@import是CSS提供的;

(2) 页面被加载的时，link会同时被加载，而@import被引用的CSS会等到引用它的CSS文件被加载完再加载;

(3) import只在IE5以上才能识别，而link是HTML标签，无兼容问题;

(4) link方式的样式的权重 高于@import的权重.

**28、position:absolute和float属性的异同**

共同点：对内联元素设置float和absolute属性，可以让元素脱离文档流，并且可以设置其宽高。

不同点：float仍会占据位置，absolute会覆盖文档流中的其他元素。

**29、介绍一下box-sizing属性？**

box-sizing属性主要用来控制元素的盒模型的解析模式。默认值是content-box。

content-box：让元素维持W3C的标准盒模型。元素的宽度/高度由border + padding + content的宽度/高度决定，设置width/height属性指的是content部分的宽/高

border-box：让元素维持IE传统盒模型（IE6以下版本和IE6~7的怪异模式）。设置width/height属性指的是border + padding + content

标准浏览器下，按照W3C规范对盒模型解析，一旦修改了元素的边框或内距，就会影响元素的盒子尺寸，就不得不重新计算元素的盒子尺寸，从而影响整个页面的布局。

**32、对BFC规范的理解？**

BFC，块级格式化上下文，一个创建了新的BFC的盒子是独立布局的，盒子里面的子元素的样式不会影响到外面的元素。在同一个BFC中的两个毗邻的块级盒在垂直方向（和布局方向有关系）的margin会发生折叠。

（W3C CSS 2.1 规范中的一个概念，它决定了元素如何对其内容进行布局，以及与其他元素的关系和相互作用。

**33、说说你对语义化的理解？**

1，去掉或者丢失样式的时候能够让页面呈现出清晰的结构

2，有利于SEO：和搜索引擎建立良好沟通，有助于爬虫抓取更多的有效信息：爬虫依赖于标签来确定上下文和各个关键字的权重；

3，方便其他设备解析（如屏幕阅读器、盲人阅读器、移动设备）以意义的方式来渲染网页；

4，便于团队开发和维护，语义化更具可读性，是下一步吧网页的重要动向，遵循W3C标准的团队都遵循这个标准，可以减少差异化。

**34、Doctype作用? 严格模式与混杂模式如何区分？它们有何意义?**

1）、<!DOCTYPE> 声明位于文档中的最前面，处于 <html> 标签之前。告知浏览器以何种模式来渲染文档。

2）、严格模式的排版和 JS 运作模式是 以该浏览器支持的最高标准运行。

3）、在混杂模式中，页面以宽松的向后兼容的方式显示。模拟老式浏览器的行为以防止站点无法工作。

4）、DOCTYPE不存在或格式不正确会导致文档以混杂模式呈现。

**35、你知道多少种Doctype文档类型？**

该标签可声明三种 DTD 类型，分别表示严格版本、过渡版本以及基于框架的 HTML 文档。

HTML 4.01 规定了三种文档类型：Strict、Transitional 以及 Frameset。

XHTML 1.0 规定了三种 XML 文档类型：Strict、Transitional 以及 Frameset。

Standards （标准）模式（也就是严格呈现模式）用于呈现遵循最新标准的网页，而 Quirks

（包容）模式（也就是松散呈现模式或者兼容模式）用于呈现为传统浏览器而设计的网页。

**36、HTML与XHTML——二者有什么区别**

区别：

1.所有的标记都必须要有一个相应的结束标记

2.所有标签的元素和属性的名字都必须使用小写

3.所有的XML标记都必须合理嵌套

4.所有的属性必须用引号""括起来

5.把所有<和&特殊符号用编码表示

6.给所有属性赋一个值

7.不要在注释内容中使“--”

8.图片必须有说明文字

**37、常见兼容性问题？**

1. png24位的图片在iE6浏览器上出现背景，解决方案是做成PNG8.也可以引用一段脚本处理.
2. 浏览器默认的margin和padding不同。解决方案是加一个全局的\*{margin:0;padding:0;}来统一。
3. IE6双边距bug:块属性标签float后，又有横行的margin情况下，在ie6显示margin比设置的大。
4. 浮动ie产生的双倍距离（IE6双边距问题：在IE6下，如果对元素设置了浮动，同时又设置了margin-left或margin-right，margin值会加倍。）
5. #box{ float:left; width:10px; margin:0 0 0 100px;}这种情况之下IE会产生20px的距离，解决方案是在float的标签样式控制中加入
6. \_display:inline;将其转化为行内属性。(\_这个符号只有ie6会识别)
7. 渐进识别的方式，从总体中逐渐排除局部。

首先，巧妙的使用“\9”这一标记，将IE游览器从所有情况中分离出来。

接着，再次使用“+”将IE8和IE7、IE6分离开来，这样IE8已经独立识别。

1. css

.bb{

background-color:#f1ee18;/\*所有识别\*/

.background-color:#00deff\9; /\*IE6、7、8识别\*/

+background-color:#a200ff;/\*IE6、7识别\*/

\_background-color:#1e0bd1;/\*IE6识别\*/

}

1. 怪异模式问题：漏写DTD声明，Firefox仍然会按照标准模式来解析网页，但在IE中会触发怪异模式。为避免怪异模式给我们带来不必要的麻烦，最好养成书写DTD声明的好习惯。现在可以使用[html5](http://www.w3.org/TR/html5/single-page.html)推荐的写法：`<doctype html>`
2. 上下margin重合问题： ie和ff都存在，相邻的两个div的margin-left和margin-right不会重合，但是margin-top和margin-bottom却会发生重合。

解决方法，养成良好的代码编写习惯，同时采用margin-top或者同时采用margin-bottom。

**38、解释下浮动和它的工作原理？清除浮动的技巧**

浮动元素脱离文档流，不占据空间。浮动元素碰到包含它的边框或者浮动元素的边框停留。

1.使用空标签清除浮动。

这种方法是在所有浮动标签后面添加一个空标签 定义css clear:both. 弊端就是增加了无意义标签。

2.使用overflow。

给包含浮动元素的父标签添加css属性 overflow:auto; zoom:1; zoom:1用于兼容IE6。

3.使用after伪对象清除浮动。

该方法只适用于非IE浏览器。具体写法可参照以下示例。使用中需注意以下几点。一、该方法中必须为需要清除浮动元素的伪对象中设置 height:0，否则该元素会比实际高出若干像素；

**39、浮动元素引起的问题和解决办法？**

浮动元素引起的问题：

（1）父元素的高度无法被撑开，影响与父元素同级的元素

（2）与浮动元素同级的非浮动元素（内联元素）会跟随其后

（3）若非第一个元素浮动，则该元素之前的元素也需要浮动，否则会影响页面显示的结构

解决方法：

使用CSS中的clear:both;属性来清除元素的浮动可解决2、3问题，对于问题1，添加如下样式，给父元素添加clearfix样式：

.clearfix:after{content: ".";display: block;height: 0;clear: both;visibility: hidden;}

.clearfix{display: inline-block;} /\* for IE/Mac \*/

清除浮动的几种方法：

1，额外标签法，<div style="clear:both;"></div>（缺点：不过这个办法会增加额外的标签使HTML结构看起来不够简洁。）

2，使用after伪类

#parent:after{

content:".";

height:0;

visibility:hidden;

display:block;

clear:both;

}

3,浮动外部元素

4,设置overflow为hidden或者auto

**40、DOM操作——怎样添加、移除、移动、复制、创建和查找节点。**

1）创建新节点

createDocumentFragment() //创建一个DOM片段

createElement() //创建一个具体的元素

createTextNode() //创建一个文本节点

2）添加、移除、替换、插入

appendChild()

removeChild()

replaceChild()

insertBefore() //并没有insertAfter()

3）查找

getElementsByTagName() //通过标签名称

getElementsByName() //通过元素的Name属性的值(IE容错能力较强，会得到一个数组，其中包 括id等于name值的)

getElementById() //通过元素Id，唯一性

**41、html5有哪些新特性、移除了那些元素？如何处理HTML5新标签的浏览器兼容问题？如何区分 HTML 和 HTML5？**

HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加。

拖拽释放(Drag and drop) API

语义化更好的内容标签（header,nav,footer,aside,article,section）

音频、视频API(audio,video)

画布(Canvas) API

地理(Geolocation) API

本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失；sessionStorage 的数据在浏览器关闭后自动删除

表单控件，calendar、date、time、email、url、search

新的技术webworker, websocket, Geolocation

移除的元素

纯表现的元素：basefont，big，center，font, s，strike，tt，u；

对可用性产生负面影响的元素：frame，frameset，noframes；

支持HTML5新标签：

IE8/IE7/IE6支持通过document.createElement方法产生的标签， 可以利用这一特性让这些浏览器支持HTML5新标签， 当然最好的方式是直接使用成熟的框架、使用最多的是html5shim框架

<!--[if lt IE 9]>

<script> src="http://html5shim.googlecode.com/svn/trunk/html5.js"</script>

<![endif]-->

32、H5 和 CSS3的新特性

H5：

①新的文档类型 <!DOCTYPE html>

②脚本和连接无需type

③语义化标签代替div，

<header>页眉，通常是一些引导和导航信息。

<nav>、

<hgroup>用于对网页或区段的标题元素（h1-h6）进行组合。

<section>定义文档中的节（section、区段），比如章节、页眉、页脚或文档中的其他部分。

<article>用来在页面中表示一套结构完整且独立的内容部分，如帖子、文章、博客、评论

<aside>用来装载非正文类的内容。例如广告，成组的链接，侧边栏等等。

<figure>规定独立的流内容（图像、图表、照片、代码等等）、

<figcaption>定义 figure 元素的标题（caption）。

<details>于描述文档或文档某个部分的细节

<summary> details 元素的标题；

<datalist>选项列表：与 input 元素配合使用，来定义 input 可能的值；

<dialog>定义一段对话；

<address>定义文章或页面作者的详细联系信息；

<keygen>给表单添加一个公钥；

<progress>定义进度条；

<time>用来表现时间或日期；

<mark> 需要标记的词或句子；

<footer>

④网页多媒体，audio和video

⑤绘制图形canvas

⑥Web存储与应用缓存

⑦跨文档消息传递XDM

⑧地理位置，Ceolocation API

⑨多线程，JS引擎可以并发的执行JS代码

⑩表单API、3DShaders(3D阴影)、针对触摸屏设备的事件、点对点网络通信

CSS3：

选择器、多列布局、多背景图、字体、RGBA和透明度、文字阴影、边框图片、边框圆角、盒阴影、盒子大小、媒体查询、语音、Word Wrap

1. 颜色：新增RGBA，HSLA模式

2. 文字阴影（text-shadow）

3. 边框： 边框圆角（border-radius）边框阴影（box-shadow ）

4. 盒子模型：box-sizing

5. 背景

background-size 设置背景图片的尺寸

background-origin 设置背景图片的原点

background-clip 设置背景图片的裁切区域，以”，”分隔可以设置多背景，用于自适应布局

6. 渐变：linear-gradient、radial-gradient

7. 过渡：transition，可实现动画

8. 自定义动画

9. 在CSS3中唯一引入的伪元素是 ::selection.

10. 媒体查询，多栏布局

11. border-image

12.2D转换：transform：translate(x，y) rotate(x，y) skew(x，y)scale(x，y)

13.3D转换

**CSS 选择符有哪些？哪些属性可以继承？优先级算法如何计算？ CSS3新增伪类有那些？**

1.id选择器（ # myid）

2.类选择器（.myclassname）

3.标签选择器（div, h1, p）

4.相邻选择器（h1 + p）

5.子选择器（ul > li）

6.后代选择器（li a）

7.通配符选择器（ \* ）

8.属性选择器（a[rel = "external"]）

9.伪类选择器（a: hover, li:nth-child）

优先级为:

!important > id > class > tag

CSS3新增伪类举例：

p:first-of-type 选择属于其父元素的首个 <p> 元素的每个 <p> 元素。

p:last-of-type 选择属于其父元素的最后 <p> 元素的每个 <p> 元素。

p:only-of-type 选择属于其父元素唯一的 <p> 元素的每个 <p> 元素。

p:only-child 选择属于其父元素的唯一子元素的每个 <p> 元素。

p:nth-child(2) 选择属于其父元素的第二个子元素的每个 <p> 元素。

:enabled :disabled 控制表单控件的禁用状态。

:checked 单选框或复选框被选中。

**42、如何实现浏览器内多个标签页之间的通信?**

调用localstorge、cookies等本地存储方式

**43、什么是 FOUC（无样式内容闪烁）？你如何来避免 FOUC？**

FOUC - Flash Of Unstyled Content 文档样式闪烁

<style type="text/css" media="all">@import "../fouc.css";</style>

而引用CSS文件的@import就是造成这个问题的罪魁祸首。IE会先加载整个HTML文档的DOM，然后再去导入外部的CSS文件，因此，在页面DOM加载完成到CSS导入完成中间会有一段时间页面上的内容是没有样式的，这段时间的长短跟网速，电脑速度都有关系。

解决方法简单的出奇，只要在<head>之间加入一个<link>或者<script>元素就可以了。

**44、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。典型用法是：

（1）变量被声明了，但没有赋值时，就等于undefined。

（2) 调用函数时，应该提供的参数没有提供，该参数等于undefined。

（3）对象没有赋值的属性，该属性的值为undefined。

（4）函数没有返回值时，默认返回undefined。

null表示”没有对象”，即该处不应该有值。典型用法是：

（1） 作为函数的参数，表示该函数的参数不是对象。

（2） 作为对象原型链的终点。

**45、new操作符具体干了什么呢?**

1、创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。

2、属性和方法被加入到 this 引用的对象中。

3、新创建的对象由 this 所引用，并且最后隐式的返回 this 。

var obj = {};

obj.\_\_proto\_\_ = Base.prototype;

Base.call(obj);

**46、js延迟加载的方式有哪些？**

defer和async、动态创建DOM方式（创建script，插入到DOM中，加载完毕后callBack）、按需异步载入js

**47、哪些操作会造成内存泄漏？**

内存泄漏指任何对象在您不再拥有或需要它之后仍然存在。

垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用数量为 0（没有其他对象引用过该对象），或对该对象的惟一引用是循环的，那么该对象的内存即可回收。

setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。

闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

**48、列举IE 与其他浏览器不一样的特性？**

IE支持currentStyle，FIrefox使用getComputStyle

IE 使用innerText，Firefox使用textContent

滤镜方面：IE:filter:alpha(opacity= num)；Firefox：-moz-opacity:num

事件方面：IE：attachEvent：火狐是addEventListener

鼠标位置：IE是event.clientX；火狐是event.pageX

IE使用event.srcElement；Firefox使用event.target

IE中消除list的原点仅需margin:0即可达到最终效果；FIrefox需要设置margin:0;padding:0以及list-style:none

CSS圆角：ie7以下不支持圆角

**49、WEB应用从服务器主动推送Data到客户端有那些方式？**

Javascript数据推送

Commet：基于HTTP长连接的服务器推送技术

基于WebSocket的推送方案

SSE（Server-Send Event）：服务器推送数据新方式

**50、对前端界面工程师这个职位是怎么样理解的？它的前景会怎么样？**

前端是最贴近用户的程序员，比后端、数据库、产品经理、运营、安全都近。

1、实现界面交互

2、提升用户体验

3、有了Node.js，前端可以实现服务端的一些事情

前端是最贴近用户的程序员，前端的能力就是能让产品从 90分进化到 100 分，甚至更好，

参与项目，快速高质量完成实现效果图，精确到1px；

与团队成员，UI设计，产品经理的沟通；

做好的页面结构，页面重构和用户体验；

处理hack，兼容、写出优美的代码格式；

针对服务器的优化、拥抱最新前端技术。

**51、一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么？**

分为4个步骤：

（1），当发送一个URL请求时，不管这个URL是Web页面的URL还是Web页面上每个资源的URL，浏览器都会开启一个线程来处理这个请求，同时在远程DNS服务器上启动一个DNS查询。这能使浏览器获得请求对应的IP地址。

（2）， 浏览器与远程`Web`服务器通过`TCP`三次握手协商来建立一个`TCP/IP`连接。该握手包括一个同步报文，一个同步-应答报文和一个应答报文，这三个报文在 浏览器和服务器之间传递。该握手首先由客户端尝试建立起通信，而后服务器应答并接受客户端的请求，最后由客户端发出该请求已经被接受的报文。

（3），一旦`TCP/IP`连接建立，浏览器会通过该连接向远程服务器发送`HTTP`的`GET`请求。远程服务器找到资源并使用HTTP响应返回该资源，值为200的HTTP响应状态表示一个正确的响应。

（4），此时，`Web`服务器提供资源服务，客户端开始下载资源。

请求返回后，便进入了我们关注的前端模块

简单来说，浏览器会解析`HTML`生成`DOM Tree`，其次会根据CSS生成CSS Rule Tree，而`javascript`又可以根据`DOM API`操作`DOM`

**52、javascript对象的几种创建方式**

1，工厂模式

2，构造函数模式

3，原型模式

4，混合构造函数和原型模式

5，动态原型模式

6，寄生构造函数模式

7，稳妥构造函数模式

**53、javascript继承的6种方法**

1，原型链继承

2，借用构造函数继承

3，组合继承(原型+借用构造)

4，原型式继承

5，寄生式继承

6，寄生组合式继承

**54、异步加载和延迟加载**

1.异步加载的方案： 动态插入script标签

2.通过ajax去获取js代码，然后通过eval执行

3.script标签上添加defer或者async属性

4.创建并插入iframe，让它异步执行js

5.延迟加载：有些 js 代码并不是页面初始化的时候就立刻需要的，而稍后的某些情况才需要的。

**55、ie各版本和chrome可以并行下载多少个资源**

IE6 两个并发，iE7升级之后的6个并发，之后版本也是6个

Firefox，chrome也是6个

**56、Flash、Ajax各自的优缺点，在使用中如何取舍？**

Flash适合处理多媒体、矢量图形、访问机器；对CSS、处理文本上不足，不容易被搜索。

-Ajax对CSS、文本支持很好，支持搜索；多媒体、矢量图形、机器访问不足。

共同点：与服务器的无刷新传递消息、用户离线和在线状态、操作DOM

**57、请解释一下 JavaScript 的同源策略。**

概念:同源策略是客户端脚本（尤其是Javascript）的重要的安全度量标准。它最早出自Netscape Navigator2.0，其目的是防止某个文档或脚本从多个不同源装载。

这里的同源策略指的是：协议，域名，端口相同，同源策略是一种安全协议。

指一段脚本只能读取来自同一来源的窗口和文档的属性。

**58、为什么要有同源限制？**

我们举例说明：比如一个黑客程序，他利用Iframe把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过Javascript读取到你的表单中input中的内容，这样用户名，密码就轻松到手了。

缺点：

现在网站的JS 都会进行压缩，一些文件用了严格模式，而另一些没有。这时这些本来是严格模式的文件，被 merge 后，这个串就到了文件的中间，不仅没有指示严格模式，反而在压缩后浪费了字节。

GET和POST的区别，何时使用POST？

GET：一般用于信息获取，使用URL传递参数，对所发送信息的数量也有限制，一般在2000个字符

POST：一般用于修改服务器上的资源，对所发送的信息没有限制。

GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值，也就是说Get是通过地址栏来传值，而Post是通过提交表单来传值。

然而，在以下情况中，请使用 POST 请求：

无法使用缓存文件（更新服务器上的文件或数据库）

向服务器发送大量数据（POST 没有数据量限制）

发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠

**59、事件、IE与火狐的事件机制有什么区别？ 如何阻止冒泡？**

1. 我们在网页中的某个操作（有的操作对应多个事件）。例如：当我们点击一个按钮就会产生一个事件。是可以被 JavaScript 侦测到的行为。

2. 事件处理机制：IE是事件冒泡、firefox同时支持两种事件模型，也就是：捕获型事件和冒泡型事件。

3. `ev.stopPropagation()`;注意旧ie的方法 `ev.cancelBubble = true`;

**60、ajax的缺点和在IE下的问题？**

ajax的缺点

1、ajax不支持浏览器back按钮。

2、安全问题 AJAX暴露了与服务器交互的细节。

3、对搜索引擎的支持比较弱。

4、破坏了程序的异常机制。

5、不容易调试。

IE缓存问题

在IE浏览器下，如果请求的方法是GET，并且请求的URL不变，那么这个请求的结果就会被缓存。解决这个问题的办法可以通过实时改变请求的URL，只要URL改变，就不会被缓存，可以通过在URL末尾添加上随机的时间戳参数('t'= + new Date().getTime())

或者：

open('GET','demo.php?rand=+Math.random()',true);//

Ajax请求的页面历史记录状态问题

可以通过锚点来记录状态，location.hash。让浏览器记录Ajax请求时页面状态的变化。

还可以通过HTML5的history.pushState，来实现浏览器地址栏的无刷新改变

**61、谈谈你对重构的理解**

网站重构：在不改变外部行为的前提下，简化结构、添加可读性，而在网站前端保持一致的行为。也就是说是在不改变UI的情况下，对网站进行优化，   
在扩展的同时保持一致的UI。

对于传统的网站来说重构通常是：

表格(table)布局改为DIV+CSS

使网站前端兼容于现代浏览器(针对于不合规范的CSS、如对IE6有效的)

对于移动平台的优化

针对于SEO进行优化

深层次的网站重构应该考虑的方面

减少代码间的耦合

让代码保持弹性

严格按规范编写代码

设计可扩展的API

代替旧有的框架、语言(如VB)

增强用户体验

通常来说对于速度的优化也包含在重构中

压缩JS、CSS、image等前端资源(通常是由服务器来解决)

程序的性能优化(如数据读写)

采用CDN来加速资源加载

对于JS DOM的优化

HTTP服务器的文件缓存

**62、HTTP状态码**

100 Continue 继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息

200 请求成功

201 请求成功并且服务器创建了新的资源

202 服务器已接受请求，但尚未处理

301 请求的网页已永久移动到新位置。

302 临时性重定向（请求的资源临时从不同的 URI响应请求）。

303 临时性重定向，且总是使用 GET 请求新的 URI。

304 自从上次请求后，请求的网页未修改过。

400 错误请求（请求的参数错误或者服务器不理解请求的语法）

401 请求未授权。

402 该状态码是为了将来可能的需求而预留的

403 禁止访问。

404 找不到如何与 URI 相匹配的资源。

500 服务器端错误。

502：网关或代理无效/无响应，网络错误

503 服务器端暂时无法处理请求（可能是过载或维护）。

**63、说说你对Promise的理解**

依照 Promise/A+ 的定义，Promise 有四种状态：

pending: 初始状态, 非 fulfilled 或 rejected.

fulfilled: 成功的操作.

rejected: 失败的操作.

settled: Promise已被fulfilled或rejected，且不是pending

另外， fulfilled 与 rejected 一起合称 settled。

Promise 对象用来进行延迟(deferred) 和异步(asynchronous ) 计算。

Promise 的构造函数

构造一个 Promise，最基本的用法如下：

var promise = new Promise(function(resolve, reject) {

if (...) { // succeed

resolve(result);

} else { // fails

reject(Error(errMessage));

}

});

Promise 实例拥有 then 方法（具有 then 方法的对象，通常被称为 thenable）。它的使用方法如下：

promise.then(onFulfilled, onRejected)

接收两个函数作为参数，一个在 fulfilled 的时候被调用，一个在 rejected 的时候被调用，接收参数就是 future，onFulfilled 对应 resolve, onRejected 对应 reject。

**64、说说你对前端架构师的理解**

负责前端团队的管理及与其他团队的协调工作，提升团队成员能力和整体效率；   
带领团队完成研发工具及平台前端部分的设计、研发和维护；   
带领团队进行前端领域前沿技术研究及新技术调研，保证团队的技术领先   
负责前端开发规范制定、功能模块化设计、公共组件搭建等工作，并组织培训。

实现一个函数clone，可以对JavaScript中的5种主要的数据类型（包括Number、String、Object、Array、Boolean）进行值复制

Object.prototype.clone = function(){

var o = this.constructor === Array ? [] : {};

for(var e in this){

o[e] = typeof this[e] === "object" ? this[e].clone() : this[e];

}

return o;

}

**65、说说严格模式的限制**

严格模式主要有以下限制：

变量必须声明后再使用

函数的参数不能有同名属性，否则报错

不能使用with语句

不能对只读属性赋值，否则报错

不能使用前缀0表示八进制数，否则报错

不能删除不可删除的属性，否则报错

不能删除变量delete prop，会报错，只能删除属性delete global[prop]

eval不会在它的外层作用域引入变量

eval和arguments不能被重新赋值

arguments不会自动反映函数参数的变化

不能使用arguments.callee

不能使用arguments.caller

禁止this指向全局对象

不能使用fn.caller和fn.arguments获取函数调用的堆栈

增加了保留字（比如protected、static和interface）

设立”严格模式”的目的，主要有以下几个：

消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;

消除代码运行的一些不安全之处，保证代码运行的安全；

提高编译器效率，增加运行速度；

为未来新版本的Javascript做好铺垫。

注：经过测试IE6,7,8,9均不支持严格模式。

**66、如何删除一个cookie**

1.将时间设为当前时间往前一点。

var date = new Date();

date.setDate(date.getDate() - 1);//真正的删除

setDate()方法用于设置一个月的某一天。

2.expires的设置

document.cookie = 'user='+ encodeURIComponent('name') + ';expires = ' + new Date(0)

<strong>，<em>和<b>，<i>标签

<strong> 标签和 <em> 标签一样，用于强调文本，但它强调的程度更强一些。

em 是 斜体强调标签，更强烈强调，表示内容的强调点。相当于html元素中的 <i>...</i>;

< b > < i >是视觉要素，分别表示无意义的加粗，无意义的斜体。

em 和 strong 是表达要素(phrase elements)。

**67、说说你对AMD和Commonjs的理解**

CommonJS是服务器端模块的规范，Node.js采用了这个规范。CommonJS规范加载模块是同步的，也就是说，只有加载完成，才能执行后面的操作。AMD规范则是非同步加载模块，允许指定回调函数。

AMD推荐的风格通过返回一个对象做为模块对象，CommonJS的风格通过对module.exports或exports的属性赋值来达到暴露模块对象的目的。

document.write()的用法

document.write()方法可以用在两个方面：页面载入过程中用实时脚本创建页面内容，以及用延时脚本创建本窗口或新窗口的内容。

document.write只能重绘整个页面。innerHTML可以重绘页面的一部分

编写一个方法 求一个字符串的字节长度

假设：一个英文字符占用一个字节，一个中文字符占用两个字节

function GetBytes(str){

var len = str.length;

var bytes = len;

for(var i=0; i<len; i++){

if (str.charCodeAt(i) > 255) bytes++;

}

return bytes;

}

alert(GetBytes("你好,as"));

**68、git fetch和git pull的区别**

git pull：相当于是从远程获取最新版本并merge到本地

git fetch：相当于是从远程获取最新版本到本地，不会自动merge

**70、说说你对MVC和MVVM的理解**

MVC

View 传送指令到 Controller

Controller 完成业务逻辑后，要求 Model 改变状态

Model 将新的数据发送到 View，用户得到反馈

所有通信都是单向的。

MVVM

Angular它采用双向绑定（data-binding）：View的变动，自动反映在 ViewModel，反之亦然。组成部分Model、View、ViewModel

View：UI界面

ViewModel：它是View的抽象，负责View与Model之间信息转换，将View的Command传送到Model；

Model：数据访问层

**71、请解释什么是事件代理**

事件代理（Event Delegation），又称之为事件委托。是 JavaScript 中常用绑定事件的常用技巧。顾名思义，“事件代理”即是把原本需要绑定的事件委托给父元素，让父元素担当事件监听的职务。事件代理的原理是DOM元素的事件冒泡。使用事件代理的好处是可以提高性能。

**72、attribute和property的区别是什么？**

attribute是dom元素在文档中作为html标签拥有的属性；

property就是dom元素在js中作为对象拥有的属性。

所以：

对于html的标准属性来说，attribute和property是同步的，是会自动更新的，

但是对于自定义的属性来说，他们是不同步的

**73、说说网络分层里七层模型是哪七层**

应用层：应用层、表示层、会话层（从上往下）（HTTP、FTP、SMTP、DNS）

传输层（TCP和UDP）

网络层（IP）

物理和数据链路层（以太网）

每一层的作用如下：

物理层：通过媒介传输比特,确定机械及电气规范（比特Bit）

数据链路层：将比特组装成帧和点到点的传递（帧Frame）

网络层：负责数据包从源到宿的传递和网际互连（包PackeT）

传输层：提供端到端的可靠报文传递和错误恢复（段Segment）

会话层：建立、管理和终止会话（会话协议数据单元SPDU）

表示层：对数据进行翻译、加密和压缩（表示协议数据单元PPDU）

应用层：允许访问OSI环境的手段（应用协议数据单元APDU）

**74、各种协议**

ICMP协议： 因特网控制报文协议。它是TCP/IP协议族的一个子协议，用于在IP主机、路由器之间传递控制消息。   
 TFTP协议： 是TCP/IP协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。   
 HTTP协议： 超文本传输协议，是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。   
 DHCP协议： 动态主机配置协议，是一种让系统得以连接到网络上，并获取所需要的配置参数手段。

**75、说说mongoDB和MySQL的区别**

MySQL是传统的关系型数据库，MongoDB则是非关系型数据库

mongodb以BSON结构（二进制）进行存储，对海量数据存储有着很明显的优势。

对比传统关系型数据库,NoSQL有着非常显著的性能和扩展性优势

与关系型数据库相比，MongoDB的优点有：   
 ①弱一致性（最终一致），更能保证用户的访问速度：   
 ②文档结构的存储方式，能够更便捷的获取数据。

**76、讲讲304缓存的原理**

服务器首先产生ETag，服务器可在稍后使用它来判断页面是否已经被修改。本质上，客户端通过将该记号传回服务器要求服务器验证其（客户端）缓存。

304是HTTP状态码，服务器用来标识这个文件没修改，不返回内容，浏览器在接收到个状态码后，会使用浏览器已缓存的文件

客户端请求一个页面（A）。 服务器返回页面A，并在给A加上一个ETag。 客户端展现该页面，并将页面连同ETag一起缓存。 客户再次请求页面A，并将上次请求时服务器返回的ETag一起传递给服务器。 服务器检查该ETag，并判断出该页面自上次客户端请求之后还未被修改，直接返回响应304（未修改——Not Modified）和一个空的响应体。

**77、什么样的前端代码是好的**

高复用低耦合，这样文件小，好维护，而且好扩展。