###浏览器的历史

\* Tim Berners-Lee发明万维网的同时发布了第一款浏览器 Nexus,也叫www浏览器

\* 由于Nexus只能显示文字, 马克安德森在伊利诺伊大学开发了第一款可以浏览图片网页的浏览器 Mosaic

\* 1994年4月,马克安德森和吉姆克拉克成立了Mosaic Communication Corporation

\* 但是由于Mosaic的版权归伊利诺伊大学所有,并且伊利诺伊大学将技术转让给了Spy Glass公司,所以在1994年11月Mosaic公司改名为Netscape Communication Corporation,浏览器改名为Netscape Navigator

\* 1996年微软买下Spy Glass公司,以Mosaic为基础开发出Internet Explorer

\* 1996年,挪威的Telenor公司开发了Opera浏览器

\* 2003年,Netscape公司倒闭,将Netscape Navigator浏览器源代码开放,Mozilla公司在Netscape Navigator浏览器的基础上开放出了FireFox

\* 2003年,苹果公司开发了safari浏览器

\* 2008年,Google发布Chrome浏览器

###JavaScript历史

\* 1995年,兰登艾奇(Brendan Eich)开发出LiveScript,并首次运行在Netscape Navigator 2上,用来完成简单的表单验证

\* 因为市场宣传需求,LiveScript与Sun公司合作,发布前将LiveScript改名为JavaScript

\* 随后在Netscape Navigator 3中发布了JavaScript1.1,同时微软在IE3中加入JavaScript的实现JScript,ScriptEase中加入JavaScript的实现CEnvi,至此浏览器上有三种不同版本的脚本语言

\* 1997年,为了统一,欧洲计算机制造商协会ECMA(European Computer Manufacturers Association）的39号技术委员会(TC39 Technical Committee#39)以JavaScript1.1为蓝本,完成了ECMA-262号标准的制定,即定义了一种名为ECMAScript的新的脚本语言标准,届时ES1诞生

\* 1998年,ES2发布,但是ES2没有任何改动,该版本主要是为了和ISO/IEC-16262保持一致

\* 1999年,ES3发布,ES3新增了正则,异常处理,控制语句等,自此ES成了一门正真的语言

\* 2000年,ES4开始准备,但是这个版本最后没有通过,但是它的大部分内容被ES6继承了

\* 2007年,ES4草案发布,ES4做了一次全面的修改,包括强类型变量,类和继承,但是由于改动太大,ECMA和各大浏览器厂商发生了严重分歧,最后ECMA开会决定,中止ES4的开发

\* 2009年12月3号,ES5发布,新增JSON对象,严格模式等,到了2011年,ES5成为了ISO国际标准

\* 2015年6月,ES6发布

\* 2016年,ES7发布

\* 2017年6月,ES8发布

###浏览器组成

\* Shell部分

\* 内核

> \* 内核的组成

>> \* 渲染引擎

>> \* JS引擎

>> \* 其他模块

> \* 主流内核介绍

>> \* Trident(IE内核)

>> \* Gecko(FireFox内核)

>> \* Webkit(Safari内核)

>> \* Blink(Chrome和Opera内核)

### JS引擎

\* 各个浏览器厂商的JS引擎去实现ES标准

\* 初期JS引擎由渲染引擎来负责,执行效率非常慢

\* 2001年,IE6将JS引擎从渲染引擎中分离出来并做了优化

\* 2008年,Google发布Chrome浏览器,并使用新的JS引擎V8,V8可以直接把JS代码转换为机器码,所以执行速度大大提高

### JS的组成

\* ECMAScript

\* DOM(Document Object Model)

\* BOM(Browser Object Model)

### JS运行在哪里

JS是一门解释性语言,运行在浏览器中,由浏览器的JS引擎负责解释运行JS代码。

### JS学什么

\* 学所有程序共同的

> 常量／变量／表达式／控制结构／数据类型／算法

\* JS这门语言独有的

> JS的内置的对象／函数

\* 前端开发相关

> 操作 DOM／BOM

### 1.写在元素标签的起始标签中

> \* 用法:把JS代码写在元素起始标签的事件属性中

> \* 缺点:重复代码多,维护困难,使用场景有限

#### 案例1:点击改颜色/免登陆提示框

\* 通过案例总结：

> JS可以修改元素的属性

> 修改元素属性的步骤

>> \* HTML页面布局

>> \* 确定修改哪些元素

>> \* 确定触发修改元素的事件，并将该事件添加在触发元素上

>> \* 编写JS代码来修改属性

> 注意点

>> \* 任何属性都可以修改

>> \* 任何标签都可以有ID属性,可以通过ID属性获取该元素

>> \* 修改属性时，HTML当中怎么写，JS当中就怎么写,但class是特例,用className

>> \* 修改样式的格式：obj.style.属性＝值

>> \* JS用style修改属性时，如果CSS的属性有"-"的话，通常把"-"去掉，然后把"-"后面的单词首字母大写

### 2.写在script标签中

> \* 把样js代码写在在script标签里面

> \* 缺点: 只适用于当前页面

> \* 注意:

>> \* 传统的做法是把script标签写在head标签里面

>> \* script标签中的type属性可以不用写, 默认就是type="text/javascript"

#### 案例2:修改多个样式

\* 通过案例总结：

> \* 可以把多行代码写在一起,然后给他起个名字,这就是函数

> \* 函数分为定义和执行,二者缺一不可

> \* 函数可以多次执行

> \* 变量相当于值的一个别名,变量出现的地方就相当于值本身

#### 案例3:网页换肤

\* 通过案例总结：

> \* 任何标签都可以加ID

> \* 任何标签的任何属性都可以修改

### 3.写在外部的单独文件中,用script标签引入

> \* 把JS写在一个外部文件里面,这个外部文件用.js作为扩展名,在需要使用的页面通过script标签的src属性引用即可

> \* 注意:

>> \* 传统的做法是把script标签写在head标签里面

>> \* script标签中的type属性可以不用写, 默认就是type="text/javascript"

>> \* 延迟属性defer,添加该属性后,脚本文件会在整个页面都解析完毕后执行

>> \* 异步属性async,添加该属性后,脚本会立即下载执行,但不妨碍页面中的其他操作

### 4. noscript标签

指定在不支持脚本的浏览器中显示替代内容

> \* 代替内容只在以下情况出现

>> \* 浏览器自身不支持脚本

>> \* 浏览器持脚本,但是被禁用了

### 语法基础

\* 区分大小写

\* 标示符(变量,函数,属性,参数的名字)的规则

> \* 第一个字符必须是字母,下划线(\_)或者美元符号($)

> \* 其他字符可以是字母,下划线(\_)或者美元符号($)或者数字

> \* 命名一般采用驼峰大小写格式,即第一个字母小写,剩下的每个单词首字母大写

> \* 不能用关键字,保留字做为标示符

\* 注释

> \* 单行注释

```javascript

// 我是单行注释

```

> \* 多行注释

```javascript

/\*

\*

\* 我是多行注释

\*

\*/

```

\* 严格模式(strict mode)

> \* 作用:告诉JS引擎切换到严格模式下解析代码,这种模式可以消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为,这是在ES5中新增的一种运行模式

> \* 使用:

>> \* 针对整个文件,将"use strict"放在脚本文件的第一行,则整个脚本都将以"严格模式"运行,如果这行语句不在第一行,则无效

>> \* 针对单个函数,将"use strict"放在函数体的第一行,则整个函数以"严格模式"运行。

> \* 注意点:

>> \* 同样的代码,在"严格模式"中,可能会有不一样的运行结果;一些在"正常模式"下可以运行的语句,在"严格模式"下将不能运行。 例如,以下代码在两种运行模式下的返回值是不同的

```javascript

function sayHi(){

alert(!this);

}

sayHi();

```

\* 每条语句以分号结尾

> 注意点:语句结尾的分号不是必须的,但建议都加上

### 程序的流程控制

#### 1. 顺序结构

> 按照代码书写的先后顺序来执行,这是程序的默认执行方式

#### 2. 分支结构

\* 语法

```javascript

if(条件){

语句

}else{

语句

}

```

\* 注意点

> 条件是一个布尔值,布尔值只有真或者假

案例:弹出菜单

#### 3. 循环结构

\* 语法1-while循环

```

初始值;

while(循环退出条件){

循环体(语句)

循环退出条件的改变

}

```

\* 语法2-for循环

```

for(初始值;循环退出条件;循环退出条件的改变){

循环体(语句)

}

```

案例1:设置多个元素的样式

案例2:全选/全不选/反选

### 函数

\* 函数的参数,参数其实就是一个占位符号,只有在调用的时候才来赋值,经常变的部分用参数

\* 匿名函数,没有名字的函数就是匿名函数

\* 案例1:改变元素的样式

\* 注意点:

> \* 在使用属性时,用[]可以代替所有的.,但是.不一定能代替[]

### 数组

数据的有序列表

\* 注意点:

> \* 可以通过下标来访问数组中的元素

> \* 数组的下标是从0开始的

> \* 可以通过数组的length属性获取到数组的长度

> \* getElementsByTagName方法获取的数据返回的就是一个数组

案例:选项卡／月历

### 1. 变量的类型

\* Undefined 未定义

> \* 未定义类型只有一个值undefined,这个值的意思是变量没有定义

> \* 未定义有两种情况,一个是真的没有定义,一个是定义了但是没有赋值

\* Null 空对象

> \* 空对象类型只有一个值null,代表一个空对象指针

\* Boolean 布尔类型

> \* 布尔类型有两个值:true和false

\* Number 数字类型

> \* 用来表示数字

> \* 可以表示整数和小数(也叫浮点数或者双精度数)

\* String 字符串类型

> \* 用单引号或者双引号括起来的字符序列

> \* 单引号和双引号表示的字符串完全一样

\* Object 对象类型

> \* 对象是一组数据和功能的集合

> \* null代表了一个空对象指针,所以用 typeof 检测时会返回 'object'

> \* 函数在ES中是对象,但函数也有一些特殊的属性,所以用 typeof 检测时会返回 'function'

> \* 数组在ES中是对象

\* 注意点:JS变量的类型由值决定,并且随着值的变化而变化

### 2. 转数值

\* 显式转换parseInt 和 parseFloat

> \* parseInt用来转换整数,parseFloat可以转换整数和小数

> \* 转换规则为

>> \* 忽略字符串前面的空格

>> \* 如果第一个字符不是数字或者负号加数字,会返回NaN(Not a Number)

>> \* 从左到右解析直到第一个非数字为止

\* isNaN

> 由于NaN和NaN并不相等,所以用isNaN函数来判断转换后的值是不是数字

\* 隐式转换

> 减法(-) 乘法(\*) 除法(/) 比较(==)

\* 案例:加法计算器

### 3. 转字符串

\* toString()

\* +""

### 4. 布尔值

> \* 真:true,非零数字,非空字符,非空对象,

> \* 假:false,数字零,空字符,null,undefined

### 5. 变量的作用域

\* 全局变量

> 定义在所有函数外面,所有的函数都可以使用的变量

\* 局部变量

> 定义在函数的内部,指定在函数内部或者函数的子函数中使用

### 6. 闭包

子函数可以使用父函数中的变量的一种现象

### 7. 变量的类型

\* 基本类型

> \* 基本类型指定的是简单的数据,包括,Undefined,Null,Boolean,Number,String

> \* 基本类型访问的变量的实际值

> \* 基本类型不能动态的添加属性

> \* 基本类型变量复制时会新建一个变量并分配空间,将变量的值分配在新的空间上

\* 引用类型

> \* 引用类型主要指保存在内存中的对象,包括对象,数组,函数

> \* 引用类型访问的是引用,引用是指向内存的一个指针

> \* 引用类型可以动态的添加属性

> \* 引用类型复制时不会新建空间,只是复制了一个新的引用

\* 引用类型

### 8. 变量命名

\* 变量名 = 类型 + 对象描述

> \* 类型指变量的数据类型,JS命名类型有

>> array 数组a

>>

>> boolean 布尔值b

>>

>> float 浮点数l

>>

>> function 函数fn

>>

>> int 整型i

>>

>> object 对象o

>>

>> regular 正则r

>>

>> string 字符串s

> \* 对象描述指对象名字全称或名字的一部分，要求有明确含义，命名要容易记忆容易理解

### 1. 算数

> 加(+), 减(-), 乘(\*), 除(/), 模(%),++,- -

>

> 案例1:隔行变色

> 案例2:秒变时间

### 2. 赋值

> =, +=, -=, \*=, /=, %=

### 3. 关系

> <, >,<=,>=,==,!=,===,!==

### 4. 逻辑

> 与(&&),或(||),否(!)

### 5. 优先级

> 括号()

### 函数参数

\* 函数的参数,参数其实就是一个占位符号.

\* 匿名函数,没有名字的函数就是匿名函数

\* 注意点:

> \* 在使用属性时,用'[]'可以代替所有的'.',但是'.'不一定能代替'[]'

>\* 案例1:改变元素的样式函数

### 函数的返回值

在函数体内通过return把函数的执行结果返回

> \* 一个函数只能有一个返回值

> \* 没有返回值或者只有return的函数返回值是undefined

> \* 函数的返回值是在调用的时候获取的

### 变量的作用域

\* 全局变量

> 定义在所有函数外面,所有的函数都可以使用的变量

\* 局部变量

> 定义在函数的内部,指定在函数内部或者函数的子函数中使用

\* 作用域的决定

> 函数执行时依赖于变量的作用域,这个作用域时在定义的时候决定的,而不时在调用的时候决定

### 闭包

子函数可以使用父函数中的变量

### 函数作用域

\* 块级作用域:在代码块(花括号中的代码)中声明的变量在代码块外不可用

\* 函数作用域:在函数内部声明的变量在函数内部以及子函数中都可以使用

\* JS没有块级作用域,而是函数作用域

\* JS中因为是函数作用域,所以有声明提升的现象,即JS函数中声明的所有变量(不涉及赋值)都被提前到函数的顶部

### 递归调用

自己调用自己的函数

> \* 案例:阶乘的计算

### 函数的不定参数

> \* 每一个函数都有一个参数数组,数组的名字是arguments

> \* 参数数组中的值由调用时传递,按照传递参数的顺序存储在数组中

> \* 案例1:求和函数

> \* 案例2:获取和设置样式函数

### 获取样式函数

\* 获取行间样式 obj.style[attr]

\* 获取非行间样式 getComputedStyle(obj,false)[attr]

### 函数参数

\* 函数的参数,参数其实就是一个占位符号,只有在调用的时候才来赋值,经常变的部分用参数

\* 匿名函数,没有名字的函数就是匿名函数

\* 注意点:

> \* 在使用属性时,用'[]'可以代替所有的'.',但是'.'不一定能代替'[]'

>\* 案例1:改变元素的样式函数

### 函数的返回值

在函数体内通过return把函数的执行结果返回

> \* 一个函数只能有一个返回值

> \* 没有返回值或者只有return的函数返回值是undefined

> \* 函数的返回值是在调用的时候获取的

### 变量的作用域

\* 全局变量

> 定义在所有函数外面,所有的函数都可以使用的变量

\* 局部变量

> 定义在函数的内部,指定在函数内部或者函数的子函数中使用

\* 作用域的决定

> 函数执行时依赖于变量的作用域,这个作用域时在定义的时候决定的,而不时在调用的时候决定

### 闭包

子函数可以使用父函数中的变量

### 函数作用域

\* 块级作用域:在代码块(花括号中的代码)中声明的变量在代码块外不可用

\* 函数作用域:在函数内部声明的变量在函数内部以及子函数中都可以使用

\* JS没有块级作用域,而是函数作用域

\* JS中因为是函数作用域,所以有声明提升的现象,即JS函数中声明的所有变量(不涉及赋值)都被提前到函数的顶部

### 递归调用

自己调用自己的函数

> \* 案例:阶乘的计算

### 函数的不定参数

> \* 每一个函数都有一个参数数组,数组的名字是arguments

> \* 参数数组中的值由调用时传递,按照传递参数的顺序存储在数组中

> \* 案例1:求和函数

> \* 案例2:获取和设置样式函数

### 获取样式函数

\* 获取行间样式 obj.style[attr]

\* 获取非行间样式 getComputedStyle(obj,false)[attr]/['widch']

### 数组的定义

\* 方法1 通过Array构造函数

```

var arr = new Array(1,2,3);

```

\* 方法2 通过数组字面量[]

```

var arr = [1,2,3]

```

\* 注意点

> \* 方法1和方法2的作用时一样的,方法2效率上更高一些

> \* 可以改变数组的length属性,改变后的值时数组的实际长度,可以通过这种方法清空数组

### 数组的操作

push(元素) 在数组末尾添加

pop() 删除数组的末尾元素

shift() 删除数组的第一个元素

unshift(元素) 把元素添加到数组的头部

splice(起始下标,删除个数,插入值1,插入值2...)

join(字符) 用指定的字符连接数组

数组1.concat(数组2) 把数组1和数组2合并生成一个新的数组

slice(起始下标,结束下标) 基于当前数组创建一个新数组

indexOf(查找项,起始下标) 从数组头部开始向后查找,返回查找到的数组的下标,如果没有找到返回-1

lastIndexOf(查找项,起始下标) 从数组未部开始向前查找,返回查找到的数组的下标,如果没有找到返回-1

### 数组的排序

\* 按字符排序

\* 指定排序函数

### 去除数组的重复元素

\* 数组是引用类型

### DOM基础

\* DOM节点介绍

\* 子节点

> \* 节点的分类

>> 文本节点

>>

>> 元素节点

> \* 获取子节点的方法

>> \* oParentNode.childNodes,返回一个数组,包含父节点中的所有文本节点和元素节点

>> \* 通过nodeType属性来判断节点的类型,1为元素节点,3为文本节点

>> \* oParentNode.children,返回一个数组, 包含父节点中的所有元素节点

\* 父节点

> \* oNode.parentNode,子节点的父节点对象

> \* oNode.offsetParent,子节点绝对定位时,获取参考的父节点对象

> \* 隐藏父元素案例

\* 首尾子节点

> \* firstChild, firstElementChild

> \* lastChild, lastElementChild

\* 兄弟节点

> \* nextSibling,nextElementSibling

> \* previousSibling,previousElementSibling

\* 操作属性

> \* 获取属性的值:getAttribute(名称),有属性获取值,没有属性返回null

> \* 设置属性:setAttribute(名称,值)

> \* 删除属性:removeAttribute(名称)

\* 通过类名获取元素集合

> \* 封装函数

\* 创建元素

> document.createElement('标签名')

\* 追加元素

> parentNode.appendChild(node)

\* 插入元素

> parentNode.insertBefore(node1,node2),在node2之前插入node1

\* 删除元素

> parentNode.removeChild(node)

> \* 删除父元素案例

### DOM 应用

\* 获取表格元素

> \* 隔行变色案例

> \* 添加删除行

> \* 搜索内容

>> \* toLowerCase()

>> \* search

>> \* split

\* 按内容排序

### 定时器

\* setInterval(fn,时间) 开启循环型定时器

\* setTimeout(fn,时间) 开启延迟型定时器

\* clearInterval(定时器) 关闭循环型定时器

\* clearTimeout(定时器) 关闭延迟型定时器

\* 注意点:

> \* 循环型定时器在定时器关闭前会一直按照设定的间隔时间执行,延迟型定时器只执行一次

> \* 定时器中指定的时间指的是该时间后把代码加入到执行队列中,如果队列空闲就立即执行

* JavaScript引擎是单线程的，强制所有的异步事件排队等待执行
* setTimeout 和 setInterval 在执行异步代码的时候有着根本的不同
* 如果一个计时器被阻塞而不能立即执行，它将延迟执行直到下一次可能执行的时间点才被执行（比期望的时间间隔要长些）
* 如果setInterval回调函数的执行时间将足够长（比指定的时间间隔长），它们将连续执行并且彼此之间没有时间间隔。

\* 案例1:数码时钟

> \* 系统时间对象 Date,方法 getHours、getMinutes、getSeconds,getFullYear(),getMonth(),getDate(),getDay()

> \* 字符串的charAt方法

\* 案例2:分类菜单

\* 案例3:焦点图1

\* 案例4:焦点图2

### 匀速动画

\* 匀速动画函数的要点

```javascript

function aninamtion(obj,attr,iTarget){

//1.一开始关闭定时器是为了防止重复触发函数时有多个定时器工作加快动画执行

//2.定时器做为对象的参数是为了多个物体动画时不相互干扰

clearInterval(obj.timer);

var iSpeed = 0;

obj.timer = setInterval(function(){

//由于getComputedStyle的返回值有单位,所以需要类型转换,由于透明度可能有小数,所以用parseFloat

var curr = parseFloat(getComputedStyle(obj,false)[attr]);

//透明度的处理以百分比的形式计算,所以乘以100

if(attr == 'opacity'){

curr = curr \* 100;

}

//根据当前值和目标值来决定速度的正负

//如果当前值小于目标值,速度为正

//如果当前值大于目标值,速度为负

if(curr <= iTarget){

iSpeed = 9;

}else{

iSpeed = -9;

}

//动画结束的条件,

//1.当前值和目标值的差不够一次动画时,即当目值和目标值的差小于速度时,动画结束并且物体直接到达目标

//2.由于速度有正负,所以用绝对值

if(Math.abs(iTarget - curr) <= Math.abs(iSpeed)){

//结束动画

clearInterval(obj.timer);

//透明度的处理

if(attr == 'opacity'){

obj.style[attr] = iTarget/100;

//其他有px单位的属性处理

}else{

obj.style[attr] = iTarget + 'px';

}

//当前值和目标值的差大于一次动画时,执行动画

}else{

//透明度的处理

if(attr == 'opacity'){

obj.style[attr] = (curr + iSpeed)/100;

//其他有px单位的属性处理

}else{

obj.style[attr] = curr + iSpeed + 'px';

}

}

},30)

}

```

\* 案例分享到

\* 模拟鼠标hover淡入淡出

\* 扩充知识点:

> Math.abs(num) 取num的绝对值

### 变速动画

\* 扩充知识点:

> Math.ceil(num) 向上取整

> Math.floor(num) 向下取整

\* 案例侧边栏固定(底部和中部)

\* 案例顶部滚出导航

> \* 扩充知识点:

>> \* window.innerWidth 窗口可视宽度

>> \* window.innerHeight 窗口可视高度

>> \* document.body.scrollTop 滚动距离,safari用

>> \* document.documentElement.scrollTop 滚动距离,chrome/ff...用

>> \* window.onresize事件,窗口大小发生变化时触发

>> \* window.onscroll事件,滚动条变化时触发

### 多物体

\* 定时器加在对象上

\* offsetWidht/offsetHeight指的是元素的宽和高,而不是内容的宽和高,所以用getComputedStyle

### 任意值

\* 对透明度单独处理

\* 扩充知识点:

> Math.round(num) 四舍五入

### 链式动画

\* 参数接受一个函数

### 多值动画

\* 参数接受一个对象,对象里指定属性和值

### 综合

### window对象

\* window代表一个浏览器对象

\* window对象是一个全局对象,因此在全局作用域中声明的变量函数会变成window对象的属性和方法

### window.open()

> 打开一个新窗口并返回新打开的窗口对象

\* window.open("about:blank","\_blank");

\* window.open("about:blank");

\* window.open("about:blank","\_self");

\* window.open("http://www.kuazhu.com");

### window.close()

\* 不能关闭用户打开的窗口

### window.navigator.userAgent

\* 当前浏览器的信息

### window.location

\* 当前页面的相关信息对象,可以读取和赋值

\* document.loaction和window.location是同一个对象

\* window.location.href 属性代表当前页面的完整url

\* window.location.assign(url),assign方法会打开url,如果把location.href或者location设置为一个url值,也会调用assign方法

\* location的其他属性

> host

> hostname

> pathname

> port

> search

### window.histroy

\* 用户上网的记录,从窗口被打开的那一刻算起

history.go(-1) == history.back()

history.go(1) == history.forward()

### 交互

var res = prompt("请输入你的姓名？");

var res = confirm("你确定删除吗?");

### 事件对象 event

\* 事件对象兼容处理

\* 事件流

> 事件冒泡

> 事件冒泡指具体元素接受事件,然后逐级向上传播事件

> 阻止事件冒泡 event.stopPropagation()

\* 案例: 下拉菜单

### 鼠标事件

\* 鼠标移动 onmousemove

\* 获取坐标

> event.clientX 和 event.clientY是鼠标的可视区坐标

> 通常情况需要加上滚动距离

\* 案例1: 随着鼠标移动的物体

\* 案例2: 随着鼠标移动的多个物体

### 键盘事件

\* 键盘按下 obj.onkeydown

\* 键盘抬起 obj.onkeyup

\* 键盘代码 event.keyCode

\* 辅助键 event.ctrlKey event.shiftKey event.altKey

\* 案例1:左右键移动物体

\* 案例2:ctrl+回车提交

\*

### 默认行为

> 默认行为是浏览器的自带行为

\* document.oncontextmenu 右键菜单

\* 阻止默认行为

> \* 方法一 return false

> \* 方法二 event.preventDefault()

\* 案例1:右键菜单

\* 案例2:只能输入数字的输入框

### 拖拽

> 原理:鼠标和物体的距离始终不变

obj.onmousedown

obj.onmouseup

> \* 基础拖拽

> \* 限制在浏览器中

> \* 限制在父容器中

> \* 磁性吸附

> \* 带框的拖拽

> \* 自定义滚动条(控制大小,控制透明度,控制文字)

### 事件

\* 事件是用户或者浏览器执行的某中动作,如click/mouseover

\* 事件处理程序是响应事件的函数,事件处理程序以on开头,如onclick/onmouseover

\* 事件绑定

> obj.addEventListener(事件名,函数,false)

> 事件名没有'on'

\* 事件移除

> obj.removeEventListener(事件名,函数,false)

> 移除事件监听只能是移除有函数名的，不能移除匿名函数的

### 对象的基本概念

\* 什么是对象

> 对象就是一个可以提供特定功能的黑盒子,使用者不用关心功能的具体实现,只要知道怎么用就可以了

> 举例:电视就是一个对象,通过遥控器或者电视面板上的按钮操作电视

\* 什么是面向对象的程序设计

> 开发具有特定功能的黑盒子

\* 对象学什么

> \* 使用对象 如:Date/Array/Math...

> \* 开发自己的对象

\* 对象的特点

> \* 封装 把具体的功能实现包起来,对外不公开

> \* 抽象 根据具体的业务需求把需要的属性和方法抽取出来

> \* 继承 子类具有父类的方法和属性,然后子类再开发自己特有的方法和属性

> \* 多态 把子类赋给父类,赋值后的父类具有子类的特性

\* 对象的组成

> \* 属性(变量)

> \* 方法(函数)

### 创建对象

\* 字面量创建

> \* 单体对象,适合整个系统只有一个对象的场景

> \* 对象属性的特征:无序唯一

>> \* 对象属性特征的应用,去除重复数组

>>> \* 把数组里面的值转换为对象的属性,由于对象的属性不能重复，所以此时对象的属性就把重复的值过滤掉

>>> \* 把对象的属性转换为数组的值

> \* js对象和json的区别

>> \* json是一种数据传输格式,js对象是类的实例,不能直接传输

>> \* json的键必须添加双引号,值不能是函数,js对象键不用加双引号,值可以是任意类型,如果js对象的属性不是一个合法的变量名时需要加双引号

>> \* json可以通过JSON.parse(jsonstr)转换为js对象,js对象可以用JSON.stringify(jsobj)转换为json

\* 构造函数创建

> \* 用系统自带的对象创建对象

>> \* 缺点:会有把系统自带的方法和属性覆盖掉的风险

> \* 用Object对象来创建对象

>> \* Object是所有对象的父类

>> \* 创建的对象方法和属性不能共用

> \* 用工厂模式的函数创建对象

>> \* 不能用new来创建对象

>> \* 方法不能被实例共享,每个对象实例都生成一个相同的方法

> \* 用new关键字创建对象 － 构造函数

>> \* new 的作用,函数会在一开始创建一个对象，并且这个对象就是this,最后会把这个this对象返回出去

>> \* 直接用new后可以把用工厂模式函数中创建对象和返回对象省略,但是方法还是不能被实例共享

>> \* 用new后,把工厂模式函数可以看做是类,类名一般首字母大写

> \* 构造函数方法共享－将构造函数里面的方法指向全局函数

>> \* 缺点1.全局函数会在其他地方调用，这时会有风险

>> \* 缺点2.如果方法太多，就需要创建大量的全局函数

>> \* 缺点3.违背面向对象的封装特性

> \* 构造原型模式

>> \* 原型 prototype 是存在于构造函数上的一个对象,所有的实例对象都可以访问原型对象

>> \* 可以在原型对象prototype添加共用的属性或者方法

>> \* 变化的属性和方法写在构造器中

> \* 动态原型模式

>> \* 可以把原型对象prototype上的定义写在构造函数中,定义之前需要唯一执行判断

### 内存分析

<img src="img/014-001.png">

### 面向对象案例:

\* 选项卡

\* 拖拽

\* 轮播图

### 正则体验

\* 从字符串中找出数字

```javascript

str.match(/\d+/g);

```

\* 替换字符

```javascript

var reg = /is/g;

str.replace(reg,"\*\*");

var reg = /\bis\b/g;

str.replace(reg,"\*\*");

```

\* 过滤关键词

```javascript

var reg = /淘宝|京东/g;

str.replace(reg,"\*\*");

```

\* 替换html标签

```javascript

var reg = /<[^<>]+>/g;

str.replace(reg,"\*\*");

```

\* 邮箱校验

```javascript

var reg = /^\w+@[0-9a-z-]+\.[a-z]+$/i;

reg.test(value)

```

\* 是否有汉字

```javascript

var reg = /[\u4e00-\u9fa5]+/g;

reg.test(value)

\* 计算汉字的长度

```javascript

function getLength(str){

return str.replace(/[\u4e00-\u9fa5]/g,"aa").length;

}

```

\* URL解析

```javascript

var reg = /(\w+):\/\/([\w\.]+)\/(\w\*)/;

str.match(reg)

```

\*\*\*\*

### 什么是正则

匹配字符串的规则

### 创建正则对象

\* 方法一(构造函数): new RegExp(规则,修正模式) 如:new RegExp('a','g')

\* 方法二(字面量): /规则/修正模式 如:var reg = /a/

### 原子(正则中的最小匹配单位)

\* 可见原子-键盘输出后可以看得见的字符,如符号,英文,数字等

> 注意,以下字符具有特殊含义,如果要使用原本的字符意思,需要在前面添加\来转义

> > . \* + ? $ ^ | \ () {} []

\* 不可见原子-键盘输出后看不见的字符,如换行(\n),回车(\r),制表符(\t),垂直制表符(\v) 换页符(\f)

### 原子筛选

\* | 竖线两边任意匹配

\* [] 括号中任意一个原子,可以写区间(闭区间),如[a-z0-9]

\* [^] 括号中任意一个原子之外的原子

\* 注意:一般情况下,正则中的一个字符对应字符串中的一个字符, 例如:/ab\t/表示字符串中连续出现字符 ab制表符

### 原子集合

\* . 除了回车和换行外的任意字符 等价[^\r\n]

\* \d 任意数字,等价[0-9], \D 任意非数字 等价[^0-9]

\* \w 任意数字,字母,下划线 等价[0-9a-zA-Z\_],\W 任意非数字,字母,下划线 等价[^0-9a-zA-Z\_]

\* \s 空白符 等价于[\t\n\f\r],\S 非空白符 等价于[^\t\n\f\r]

### 量词(限定前面原子出现的次数)

\* {n} 前面原子恰好出现n次

\* {n,} 前面原子最少出现n次

\* {n,m} 前面原子最少出现n次,最多出现m次

\* \+ 前面原子最少出现1次 等价{1,}

\* ? 前面原子最少出现0次,最多出现一次 等价{0,1}

\* \\* 前面原子最少出现0次 {0,}

### 匹配模式

\* 贪婪模式-尽可能多的匹配(默认)

```javascript

'12345'.replace(/\d{2,4}/,'X');

"X5"

```

\* 非贪婪模式-尽可能少的匹配(在量词后添加?)

```javascript

'12345'.replace(/\d{2,4}?/,'X');

"X345"

```

### 边界控制

\* ^ 匹配字符串开始的位置,限制必须以其后面的原子开头，前面不能有任何东西

\* $ 匹配字符串结尾的位置，限制必须以其前面的原子结束，后面不能有任何东西

\* \b 单词边界

\* \B 非单词边界

\* () 把括号里面的正常表达匹配的结果当作一个原子来看,起到分组的效果,分组后可以通过反向引用来取得分组后的值,反向引用从$1开始

```javascript

//匹配四个一个小写字母加一个数字的字符串

'a1b2c3d4aaaa'.replace(/([a-z]\d){4}/,"\*");

"\*aaaa"

//将年月日替换成月日年

'2018-05-09'.replace(/(\d{4})-(\d{2})-(\d{2})/,'$2/$3/$1');

"05/09/2018"

```

### 修正模式

\* i(ignoreCase) 忽略大小写,默认区分大小写,可以通过正则对象上的ignoreCase属性查看

\* g(global) 匹配所有的,默认只找到第一个,可以通过正则对象上的global属性查看

### 前瞻

\* 正则从文本头部向文本尾部解析,文本尾部成为"前",前瞻就是向前(文本尾部)检查是否符合断言

\* 正向前瞻 exp(?=assert) 匹配到exp的内容后再看前面的内容符合不符合assert,符合的话就匹配exp的内容

```javascript

//字符后面是数字的字符

'a2\*3b&'.replace(/\w(?=\d)/g,'X')

X2\*3b&

```

\* 负向前瞻 exp(?!assert) 匹配到exp的内容后再看前面的内容符合不符合assert,不符合的话就匹配exp的内容

```javascript

//字符后面不是数字的字符

'a2\*3b&'.replace(/\w(?!\d)/g,'X')

"aX\*XX&"

```

### 正则对象的方法

\* 正则对象属性

> lastIndex 当前正则匹配内容的最后一个字符的下一个位置

> source 正则的文本字符串

\* RegExp.prototype.test(str);

> test方法测试字符串是否有匹配到正则的字符串,有返回true,没有返回false

> 如果全局匹配时,每一次匹配后正则对象上的lastIndex属性会更改,改为匹配内容的最后一个字符的下一个位置,所以结果会不可靠,一般使用test方法时建议不要全局匹配

> 如果非全局匹配时,lastIndex始终是0,也就是说始终从第0个元素开始查找

```javascript

var reg = /\w/g;

console.log("0::",reg.lastIndex);//0

console.log("1::",reg.test('ab'));//true

console.log("2::",reg.lastIndex);//1

console.log("3::",reg.test('ab'));//true

console.log("4::",reg.lastIndex);//2

console.log("5::",reg.test('ab'));//false

console.log("6::",reg.lastIndex);//0

```

\* RegExp.prototype.exec(str);

> exec方法对字符串执行搜索,并更新正则对象上的lastIndex属性,如果没有匹配的返回null,有的话返回一个数组:

>> \* 数组的第0个元素是匹配的内容,如果有分组,从下标是1开始是对应分组的内

>> \* index 是匹配内容第一个字符的位置

>> \* input 是被搜索的字符串

> \* 如果非全局匹配时,lastIndex始终是0,也就是说始终从第0个元素开始查找

```javascript

var reg = /(\d{4})-(\d{2})-(\d{2})/g;

var str = 'a2018-05-10b';

console.log(reg.lastIndex);//0

console.log(reg.exec(str));//["2018-05-10", "2018", "05", "10", index: 1, input: "a2018-05-10b", groups: undefined]

console.log(reg.lastIndex);//11

console.log(reg.exec(str));//null

console.log(reg.lastIndex);//0

console.log(reg.exec(str));//["2018-05-10", "2018", "05", "10", index: 1, input: "a2018-05-10b", groups: undefined]

```

### 字符串方法中使用正则

\* String.prototype.search(reg)

> \* 从字符中检索于正则相匹配的字符串,返回第一个匹配结果的index,查不到返回-1

> \* search方法不执行全局匹配,它会忽略g,并且总是从字符串开始检索

\* String.prototype.math(reg)

> \* 非全局匹配的结果和exec一样

> \* 全局匹配,会找到字符串中的所有匹配字符串,如果没有匹配的话返回null,有的话返回一个数组:

>> \* 数组中的每一项就是一个匹配结果,数组中没有index和input等属性

>> \* 全局匹配不需要分组

\* String.prototype.split(reg)

> 根据正则匹配的内容把字符串分隔成数组

```javascript

'a1b2c3d'.split(/\d/);

["a", "b", "c", "d"]

```

\* String.prototype.replace(reg,str2)

> 把字符串中匹配到正则的子串用str2进行替换

\* String.prototype.replace(reg,function(匹配结果,index,原值))

> 把字符串中匹配到正则的子串传递到回调函数进一步处理,用函数的返回值来替换匹配项

```javascript

var str = 'a1b2c3d4'.replace(/\d/g,function(match,index,origin){

return parseInt(match)+1;

})

```

### String 对象方法

\* 根据位置返回字符 str.charAt(index)

> \* 位置在0和str.length-1之间,如果不在返回空字符

> \* 一个汉子是一个字节

\* 根据位置返回字符的编码 str.charCodeAt(index)

> \* 位置在0和str.length-1之间,如果不在返回空字符

> \* 一个汉子是一个字节

> \* charCodeAt返回的是字符的Unicode码

> \* Unicode码是统一的编码,Unicode码包含了ASCII码(0-127)

\* 根据字符返回位置 str.indexOf(str1) / str.lastIndexOf(str1)

> \* indexOf 从前向后找,找不到返回-1

> \* lastIndexOf 从后向前找,找不到返回-1

\* URI 编码 encodeURIComponent(str)

\* URI 解码 decodeURIComponent(str)

\* 连接字符串 str1.concat(str2)

\* 截取字符串 slice(起始下标,结束下标),下标可以是负数,负数表示从尾部开始计算

\* 截取字符串 substr(起始下标,长度),下标可以是负数,负数表示从尾部开始计算

\* 截取字符串 substring(起始下标,结束下标),下标不可以是负数

\* 把字符串转换为小写 toLowerCase()

\* 把字符串转换为大写 toUpperCase()

### 获取表单元素

\* document.forms 代表页面所有的表单集合

\* document.forms[0].elements 代表某一个表单中的所有表单元素集合

\* 获取一个表单中的元素的方法

> 表单元素集合[下标]

>

> 表单元素集合["元素名"]

>

> 表单元素集合.元素名

>

> 表单.元素名

>

> 同名的表单元素返回一个节点集合

\* 表单元素.form 代表表单元素所在的form

### 表单元素的属性

\* 禁用表单元素

> \* 表单元素.disabled = true;

>> 针对所有表单元素

>>

>> 值不会被提交

> \* 表单元素.readOnly = true;

>> 只针对input(text/password)/textarea

>> 值会被提交

\* 复选框选中属性 checked

> \* 设置选中 aCheckBox[0].checked = true

\* 单选框选中属性 checked

> \* 设置选中 aRadio[0].checked = true

\* 下拉框选中属性 selected

> \* 设置某个选项选中 oSelect.options[i].selected = true

> \* 通过选中属性获取所有选中项

```

function getSelectedElements(oSelect){

var result = [];

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

if(oSelect.options[i].selected){

result.push(oSelect.options[i]);

}

}

return result;

}

```

### 表单元素方法

\* 设置焦点 表单元素.focus();

\* 移除焦点 表单元素.blur();

### 表单的提交和重置

\* HTML按钮默认提交

> input type="submit"

> input type="image"

> button type="submit"

\* 表单提交事件 表单.onsubmit

> \* 表单被提交,数据没有发送前触发

> \* 可以在该事件中通过阻止默认行为来阻止表单的提交

\* 给普通按钮绑定事件,调用 form的submit()方法提交

> form的submit()方法不会触发form.onsubmit事件

\* HTML按钮默认提交重置

> input type="reset"

> button type="reset"

\* 表单重置事件 表单.onrest

> \* 事件在表单重置前触发

> \* 可以通过阻止默认行为来阻止重置

\* 给普通按钮绑定事件,调用 form的reset()方法重置

> form的reset()方法不会触发form.onrest事件

JS进阶

### 语法错误(SyntaxError)

> 不符合JS语法的错误,控制台会出现错误并且有行号提示(行号不一定准备)

\* 常见语法错误

> \* 中文输入指令

> \* 少打,漏打,多打

> \* 变量名不合法

> \* 语句写错

\* 注意点

> \* 控制台中的错误提示中一个token就是指JS中的一个符号,例如分号,var关键字

### 运行时错误(Runtime Error)

> JS无语法错误,在运行时发生的错误

\* ReferenceError 变量引用异常触发的错误

\* TypeError 类型使用时错误

\* RangeError 递归爆栈时触发

### 逻辑错误

> 处理结果不符合预期

\* 借助控制太单步跟踪

> \* 在程序需要开始跟踪的地方打断点

>> \* 在程序中需要开始跟踪的地方添加debugger关键字

>> \* 在控制台的source面板中单击需要跟踪程序的行号

> \* Watch 需要跟踪的变量

>> \* 可以Watch一个表达式

>> \* 可以在控制台中更改变量的值

> \* Resume script execution 按钮,执行脚本到下一个断点的位置

> \* Step over next function call,跳过函数的内部执行,把函数当做一个语句来看

> \* Step into next function call,进入到函数的内部执行

> \* Step out current function ,跳出当前的到函数内部执行

> \* Deactivate breakpoints/Activate breakpoints,关闭和打开断点

> \* Pause on exceptions,异常处停止

>> \* 如果选上 Pause on caught exceptions的话会在try语句中停止

### 主动触发异常

在程序中如果有些不可避免的错误发生时,可以主动的抛出一个异常,然后用try-catch语句来捕获异常

\* 抛出异常

```javascript

throw new Error('error msg');

```

\* 捕获异常

```

try{

...//有可能出现异常的代码

}catch(e){

...//异常处理

}finally{

...//不管有没有异常都会处理,例如清理工作

}

### 闭包的使用

\* 使用函数内部的变量,将值保存在内存中

> 让函数返回一个引用了函数内部变量的函数

\* 封闭作用域,匿名函数自我调用解决变量作用域,不会污染全局变量

\* 封闭作用域传递参数,解决i值问题

> \* 子函数会使用作用域链中最近的变量,如果整个作用域中都没有定义,则报错

> \* 父函数不能访问子函数中定义的变量

\* 函数节流

> 函数节流举例

```javascript

function throttle(fn,delay){

var timer = null;

return function(){

clearTimeout(timer);

timer = setTimeout(fn,delay);

}

}

window.onresize = throttle(function(){

console.log(window.innerWidth);

},400)

```

\* 利用闭包和原型构建对象

### 什么是设计模式

设计模式（Design Pattern）是一套被反复使用、多数人知晓的、经过分类的、代码设计经验的总结。

### 观察者模式(发布订阅者模式)

\* 发布者维护一个数组,该数组用来存储所有关注者对象

\* 发布者提供一个订阅者可以加入数组的方法

\* 发布者发布消息时循环遍历订阅者数组，调用订阅者的监听方法

\* 订阅者在创建时调用发布者的加入数组方法

\* 订阅者需要提供监听方法供发布者调用传递消息

案例1:发布作业

案例2:汇率计算

### 单例模式

保证一个类仅有一个实例,并且提供一个访问它的全局访问点

案例1:皇帝上朝

案例2:惰性登录框

\* 遍历

```javascript

arr.forEach(function(value,index){

console.log(value);

})

```

> 遍历不会生成新的数组,没有返回值

\* 映射

```javascript

var newArr = arr.map(function(value,index){

return value\*10;

})

```

> 遍历数组,把函数的每一次返回值映射成新的数组并返回

\* 过滤

```javascript

var newArr = arr.filter(function(value,index){

return value > 10;

})

```

> 遍历数组,把函数的每一次返回值为真的原数组中的值组成新的数组并返回

\* 全真判断

```javascript

var bRes = arr.every(function(value,index){

return value > 10;

})

```

> 遍历数组,所有值为真的情况下返回真,否则返回假

\* 有真判断

```javascript

var bRes = arr.some(function(value,index){

return value > 10;

})

```

> 遍历数组,有一个值为真的情况下返回真,否则返回假

### 什么是HTTP

HTTP(HyperText Transfer Protocol)超文本传输协议

### 什么是协议

协议就是一套规则，用来规范/约束某一类事物

例如租房协议/买卖协议/离婚协议

HTTP协议就是用来规范客户端(浏览器)和服务端通信规则的

> 常见的网络协议还有很多，如

> \* FTP(File Transfer Protocol)文件传输协议

> \* POP3(Post Office Protocol Version 3)邮局协议-版本3

> \* SMTP(Simple Mail Transfer Protocol)简单邮件传送协议

> \* TCP/IP(Transmission Control Protocol/Internet Protocol)传输控制协议/Internet协议

### 什么是HTTPS

HTTPS（全称：Hyper Text Transfer Protocol over Secure Socket Layer）,安全套接字层超文本传输协议HTTPS。为了数据传输的安全，HTTPS在HTTP的基础上加入了SSL协议，SSL依靠证书来验证服务器的身份，并为浏览器和服务器之间的通信加密。

### OSI(Open System Interconnection)开放式系统互联的七层架构

\* 1 物理层(设备之间的比特流传输,物理接口)

\* 2 数据链路层(成帧,用MAC地址访问媒介)

\* 3 网络层(写入IP地址(数据源和数据目标的地址),选路)

\* 4 传输层(确定是用TCP协议传输还是UDP协议传输)

> \* 传输控制协议TCP可靠但是慢

> \* 用户数据协议UDP不可靠但是快

\* 5 会话层(安全协议,目录访问协议)

\* 6 表示层(转码/解码,加密/解密)

\* 7 应用层(提供应用数据,各种应用协议,如HTTP,FTP,SMTP...)

> 发送从应用层开始向下传输,接受从物理层开始向上传输

### TCP/IP 4层模型

\* 网络接口层(对应7层中的物理层和数据链路层)

\* 网际互联层(对应7层中的网络层)

\* 传输层(对应7层中的传输层)

\* 应用层(对应7层中的应用层,表示层和会话层)

### 访问页面流程详解

\* 客户在浏览器输入网址或者刷新页面

\* 域名解析(以http://www.kuazhu.com/为例)

1 浏览器搜索自身的DNS缓存

查看浏览器缓存 chrome://net-internals/#dns

2 如果浏览器缓存没有或者过期，就去操作系统自身的DNS缓存查找

3 读取本地的HOST文件

windows：C:\Windows\System32\drivers\etc

OS:/etc/hosts

4 向宽带运营商服务器发起域名解析请求,宽带运营商服务器查看自身的缓存

5 运营商服务器发起一个迭代的DNS解析请求

5.1 找根域的服务器

5.2 根域服务器里面有com域的地址,根域让去找com域的服务器

5.3 com域服务器里面有kuazhu.com域的IP地址,com域服务器让去找kuazhu.com域的服务器

5.4 kuazhu.com域的服务器(一般是域名注册商)里面有对应的IP地址和域名的映射,找到后将IP地址发给运营商服务器

5.5 运营商服务器把结果返回给客户操作系统,同时操作系统将其缓存起来,

操作系统把结果返回给浏览器

\* 浏览器获取IP地址后,发起TCP/IP 三次握手

第一次握手：客户端发送syn包（syn=j）到服务器，并进入SYN\_SENT状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=k+1），此包发送完毕，客户端和服务器进入ESTABLISHED（TCP连接成功）状态，完成三次握手。

完成三次握手，客户端与服务器开始传送数据

\* TCP/IP连接建立后，开始发送HTTP请求

浏览器发送请求行到服务器(请求行包括请求方法 uri 协议/版本等)

浏览器发送请求头到服务器,结束后发送一个空白行告诉服务器不再有请求头(请求头包含许多有关的客户端环境信息)

浏览器发送请求体到服务器(请求体主要是请求参数)

服务器应答处理并且返回响应行(响应行包括协议/版本 状态码 状态消息)

服务器返回响应头,结束后发送一个空白行告诉客户端不再有响应头(响应头包含许多有关的服务端环境信息)

服务器返回响应体(响应体主要是返回的数据)

关闭TCP连接

\* 浏览器渲染服务器返回的内容

\* HTTP一旦请求完成,连接就会断开,因此连接是没有记忆的,无状态的

### 在浏览器中看请求响应的信息-Timing

Queued at

添加到处理队列的时间

Started at

开始处理的时间

Stalled 浏览器要发出请求到这个请求可以发出的等待时间，一般是代理协商、以及等待可复用的TCP连接释放的时间，不包括DNS查询、建立TCP连接等时间等

DNS Lookup

执行DNS查找时间

Initial connection

建立TCP连接的时间，就相当于客户端从发请求开始到TCP握手结束这一段，包括DNS查询+Proxy时间+TCP握手时间。

Request sent:请求的第一个字节发出前到最后一个字节发出后的时间

Waiting：请求发出后到收到响应的第一个字节所耗费的时间，包括数据的传输和服务器对请求的处理时间

Content Download:收到响应的第一个字节开始到收到响应的最后一字节结束所耗费的时间

### HTTP的请求方法

GET方法

用于获取数据

请求是参数的大小有限制(400k)

参数以及参数的值会显示在地址栏中，安全性极低

POST方法

用于提交数据(表单的提交)

请求是参数的大小基本没有限制

参数不会显示在地址栏中，相对安全

### 什么是URL?

uniform resource locator 统一资源定位器

可以理解为地址栏中的网址

### URL的格式

完整格式：https://www.kuazhu.com:443/courseList

协议://域名:端口号/资源文件

### 常见状态码

1xx-信息提示

100 - 继续。

101 - 切换协议。

2xx - 成功

这类状态代码表明服务器成功地接受了客户端请求。

200 - 确定。客户端请求已成功。

201 - 已创建。

202 - 已接受。

203 - 非权威性信息。

204 - 无内容。

205 - 重置内容。

206 - 部分内容。

207 - 多状态 (WebDAV)。

3xx - 重定向

客户端浏览器必须采取更多操作来实现请求。例如，浏览器可能不得不请求服务器上的不同的页面，或通过代理服务器重复该请求。

301 - 已永久移动

302 - 对象已移动。

304 - 未修改。

307 - 临时重定向。

4xx - 客户端错误

发生错误，客户端似乎有问题。例如，客户端请求不存在的页面，客户端未提供有效的身份验证信息。

400 - 错误的请求。

401 - 访问被拒绝。IIS 定义了几个不同的 401 错误，用于指示更为具体的错误原因。这些具体的错误代码在浏览器中显示，但不在 IIS 日志中显示：

401.1 - 登录失败。

401.2 - 服务器配置导致登录失败。

401.3 - 由于 ACL 对资源的限制而未获得授权。

401.4 - 筛选器授权失败。

401.5 - ISAPI/CGI 应用程序授权失败。

401.7 – 由 Web 服务器上的 URL 验证策略拒绝访问。这个错误代码为 IIS 6.0 所专用。

403 - 禁止访问：IIS 定义了几个不同的 403 错误，用于指示更为具体的错误原因：

403.1 - 执行访问被禁止。

403.2 - 读访问被禁止。

403.3 - 写访问被禁止。

403.4 - 要求 SSL。

403.5 - 要求 SSL 128。

403.6 - IP 地址被拒绝。

403.7 - 要求客户端证书。

403.8 - 站点访问被拒绝。

403.9 - 用户数过多。

403.10 - 配置无效。

403.11 - 密码更改。

403.12 - 拒绝访问映射表。

403.13 - 客户端证书被吊销。

403.14 - 拒绝目录列表。

403.15 - 超出客户端访问许可。

403.16 - 客户端证书不受信任或无效。

403.17 - 客户端证书已过期或尚未生效。

403.18 - 在当前的应用程序池中不能执行所请求的 URL。这个错误代码为 IIS 6.0 所专用。

403.19 - 不能为这个应用程序池中的客户端执行 CGI。这个错误代码为 IIS 6.0 所专用。

403.20 - Passport 登录失败。这个错误代码为 IIS 6.0 所专用。

404 - 未找到。

404.0 -（无） – 没有找到文件或目录。

404.1 - 无法通过请求的端口访问网站。

404.2 - Web 服务扩展锁定策略阻止本请求。

404.3 - MIME 映射策略阻止了此请求。

405 - 用来访问本页面的 HTTP 谓词不被允许（方法不被允许）

406 - 客户端浏览器不接受所请求页面的 MIME 类型。

407 - 要求进行代理身份验证。

412 - 前提条件失败。

413 – 请求实体太大。

414 - 请求 URI 太长。

415 – 不支持的媒体类型。

416 – 无法满足请求的范围。

417 – 执行失败。

423 – 锁定的错误。

5xx - 服务器错误

服务器由于遇到错误而不能完成该请求。

500 - 内部服务器错误。

500.12 - 应用程序正忙于在 Web 服务器上重新启动。

500.13 - Web 服务器太忙。

500.15 - 不允许直接请求 Global.asa。

500.16 – UNC 授权凭据不正确。这个错误代码为 IIS 6.0 所专用。

500.18 – 无法打开 URL 授权存储库。这个错误代码为 IIS 6.0 所专用。

500.19 - 此文件的数据在元数据库中配置不正确。

500.100 - 内部 ASP 错误。

501 - 页眉值指定了未实现的配置。

502 - Web 服务器用作网关或代理服务器时收到了无效响应。

502.1 - CGI 应用程序超时。

502.2 - CGI 应用程序出错。

503 - 服务不可用。这个错误代码为 IIS 6.0 所专用。

504 - 网关超时。

505 - HTTP 版本不受支持。

### 服务器

安装了Web服务器软件的电脑就可以提供管理网站的服务,我们称之为Web服务器。

常见的Web服务器软件有:Apache,IIS,Tomcat,Nginx,NodeJs等

### NodeJs 初识

\* 官网 https://nodejs.org/ 下载安装,注意要下载LTS版本的

\* NodeJs安装完成后,去终端用node -v 查看,如果有版本号说明安装成功

\* 如果node命令找不到,Windows的用户注意环境变量的配置

\* NodeJs安装的同时会安装npm(node package manager)

### ajax请求流程：

\* 客户端创建一个ajax对象

\* 用ajax对象和服务端建立连接

\* 用ajax对象发送请求到服务端

\* 用ajax对象接收服务端返回的数据

```javascript

function ajax(method,url,data,fnSucc,fnFaild){

// 1.创建一个ajax对象

var oAjax = new XMLHttpRequest();

//post方法处理

if(method.toLowerCase() == 'post'){

oAjax.open(method,url,true);

//设置请求头,该方法必须在open和send中间调用

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

//post发送的数据

oAjax.send(data);

}else{

if(data){

url = url+"?"+data;

}

// 2.用ajax对象建立连接

// oAjax.open(请求方法，请求地址，是否异步)

oAjax.open(method,url,true);

// 3.用ajax对象发送请求

oAjax.send();

}

// 4.用ajax对象接收数据

oAjax.onreadystatechange = function(){

// console.log(oAjax.readyState);

// 0 还没有初始化

// 1 已经初始化，但是还没有调用open方法

// 2 send方法已经调用

// 3 返回了部分数据

// 4 数据完全返回

if(oAjax.readyState == 4){//请求完成，表明所有的数据已经返回，但并不代表成功

if(oAjax.status == 200){//成功

fnSucc(oAjax.responseText);

}else{//失败

if(fnFaild){

fnFaild(oAjax.status);

}

}

}

}

}

```

### 跨域方案

\* 请求端的协议,域名,端口和服务器的协议,域名,端口有一个不一致就会发生跨域

\* CORS(Cross-origin resource sharing)跨域方案

> 服务器端设置响应头

```javascript

res.setHeader("Access-Control-Allow-Origin","\*")

```

> 注意:

> \* Access-Control-Allow-Origin设置可以跨域的域名

> \* \\*代表所有

> \* 一次只能只能设置一个值

\* 简单请求同时满足条件

> \* 请求方法必须是 HEAD,GET,POST之一

> \* 请求头中的字段仅限于 Accept,Accept-Language,Content-Language,Last-Event-ID,Content-Type

> \* Content-Type 的值仅限于application/x-www-form-urlencoded,multipart/form-data,text/plain

\* 复杂请求

不满足简单请求的时候就是复杂请求,复杂请求会先发一次OPTIONS方法进行预检(preflight)请求

例如设置如下请求头后该请求就会变为复杂请求

oAjax.setRequestHeader('abc','abc');

需要在服务器端设置如下响应头才能实现请求

```javascript

res.setHeader("Access-Control-Allow-Headers","abc");

```

> Access-Control-Allow-Headers 表示服务端接受的跨域请求的字段

\* 服务器端响应头Access-Control-Expose-Headers

> Access-Control-Expose-Headers表示允许客户端通过getResponseHeader方法获取的字段

> CORS方式下默认只能获取6个基础字段,Cache-Control,Content-Language,Content-Type,Expires,Last-Modified,Pragma