JavaScript

主讲:石小俊

一.介绍JavaScript

1.什么是JavaScript

与java毫无关系

网景公司开发的一款客户端脚本语言

不需要编译

不需要安装环境

只要浏览器

实现页面的行为,即功能

简称:`js`

2.JavaScript作用

- 在客户端动态的生成页面
- 在客户端做数据格式的校验
- 发送Ajax请求

二.引入方式

JavaScript的引入方式类似于css

1.行内式

```
<!-- 在js中,当遇到双引号中需要使用双引号时,使用单引号代替 -->
<input type="button" value="行内式" onclick="javascript:alert('Hello JavaScript!')"/>
```

2.内联方式

在html中使用`<script>`标签

在`<script>`标签中编写JavaScript代码

`<script>`标签可以存在于html文档中的任意位置

```
<script>
    function f1(){
        alert("Hello 内联");
    }
</script>

<input type="button" value="内联方式" onclick="f1()">
```

3.外部引入

定义一个`*.js`的文件

在该文件中编写js函数

在当前的html文档中通过`<script>`引入外部的js文件

```
function f2(){
    alert("这是外部引入的js的函数");
}
```

```
<script type="text/javascript" src="test.js"></script>
<input type="button" value="外部引入" onclick="f2()"><br/>
```

注意点:

- 当`<script>`标签使用了src属性之后
- 标签体中不可以出现js代码

错误用法:

```
<script type="text/javascript" src="test.js">
    function f3(){
        alert("f3");
    }
</script>
```

三.JavaScript数据类型

1.变量的用法

- 先声明,后赋值
  - var num;
  - num = 10;
  - num = "admin"
- 声明的同时赋值
  - var num = 10;
- 不声明直接赋值
  - num = 10;

    - 不建议使用

2.变量的特点

- js是一个弱类型的语言
    - 所谓的弱类型是相对于强类型来说的
    - 所谓的强类型,类似于java,定义变量的时候必须指定变量的具体类型
    - 且在定义了之后,变量的类型无法转变
    - 弱类型表示变量的类型是比较模糊的，随时可以发生变化
    - 在js中,变量的类型是在最终运行完成之后才确定的
- es6之前,js是没有作用域的概念的
    - 在es6之后,引入了作用域的概念

3.数据类型的种类

- 基本数据类型
    - number:数值类型,包含整数与小数
    - string:字符串
    - boolean:布尔类型
- 非正常类型
    - null:空类型
    - undefinded:未定义
    - NaN:Not a Number,非数字类型
- 复合数据类型
    - Array:数组类型
    - Date:日期类型
    - function:函数类型
    - object:对象类型

判断当前变量的类型的方法

- `typeof 变量名`
- `typeof(变量名)`

4.`let`与`var`的区别

4-1 介绍`let`

let是ES6后支持的语法

es6是2015年发布的一个JavaScript的版本

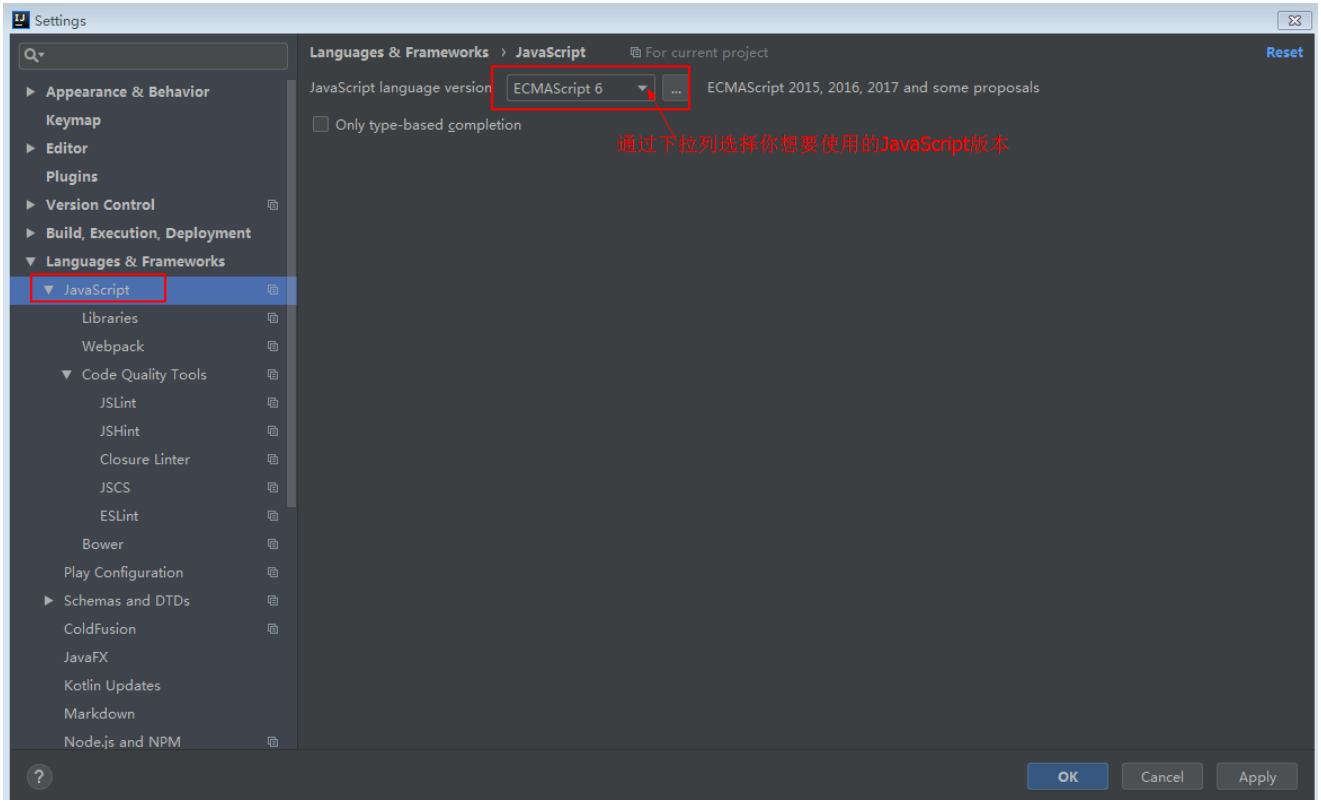
又称之为es2015

虽然后面也发布了es7、es8

一般情况下,我们将es6后面的版本都称之为es6

4-2 idea中改变当前JavaScript版本

File-->settings-->Languages & Frameworks-->JavaScript



4-3 `let`特点

`let`的行为与java中变量的行为基本一致

- 变量名不可以重复
- 存在作用域
- 不存在变量提升

4-4 `var`特点

- 变量名可以重复
- 不存在作用域
- 存在变量提升

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>let与var</title>
    <script>
        function f1(){
            var num = 10;
            var num = 10;
            let s = 1;
            // let s = 2;
            for(let i = 0; i < 10; i++){

            }
            alert(i);

        }
        function f2(){
            // var a = 2;
            //console.log()--表示在浏览器的控制台打印信息
            // console.log("这是在浏览器控制台");
            //如果单独打印一个未定义的变量值,控制台会出错
            // console.log(a);

            //而如果在打印的变量后面加上一个定义变量的数据
            //则不会产生报错,只是值属于未定义
            // var a = 22;


            console.log(a);
            var a = 22;
            //因此上述两行代码相当于如下写法
            //这就是变量提升
            //将变量的声明提升到最上方
            // var a;
            // console.log(a);
            // a = 22;

            console.log(b);
            let b = 22;
            //此处不存在变量提升,因此其出现了报错

        }
    </script>
</head>
<body>
    <a href="javascript:;" onclick="f1()">测试1</a><br/>
    <a href="javascript:;" onclick="f2()">测试2</a><br/>
</body>
</html>
```

5.变量的运算

5-1 基本运算

> +、-、*、/、=
>
> +=、-=、*=、/=、==
>
> ++、--
>
> %
>
> && || & |
>
> 三目运算

5-2 js特有运算符

5-2-1 `===`

恒等于

在js中,使用`==`进行比较的时候,只比较值,不考虑类型

但是,在某些时候,我们变量进行比较的时候,需要考虑当前的类型

此时使用`===`

5-2-2 `**`

es新语法

`a**b`表示a的b次方

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>变量的运算</title>
    <script>
        function f1(){
            var s1 = 3;
            var s2 = 2;
            console.log("s1+s2="+(s1+s2));
            console.log("s1-s2="+(s1-s2));
            console.log("s1*s2="+(s1*s2));
            console.log("s1/s2="+(s1/s2));
            console.log("s1%s2="+(s1%s2));
            console.log(s1==s2?"true":"false");
            console.log(2=="2"?"true":"false");
            if(2 == "2"){
                console.log(true);
            }else{
                console.log(false);
            }
        }
        function f2(){
            if(2 === "2"){
                console.log(true);
            }else{
                console.log(false);
            }
            var num = 2**4;
            console.log(num)
        }
    </script>
</head>
<body>
<input type="button" value="基本运算" onclick="f1()">
<input type="button" value="js特有运算符" onclick="f2()">
</body>
</html>
```

6.string

在js中,可以使用大部分java中字符串的方法

- length属性
  - 获取字符串的长度
- toUpperCase()
  - 将字符串的所有字符转换为大写
- toLowerCase()
  - 将字符串的所有字符转换为小写
- trim()

    - 去除字符串首尾两端的空格
- parseInt()
    - 将一个字符串转换为数字
    - 只能转换为整数
    - 转换规则:
        - 如果当前字符串由纯数字组成
            - 则直接转换为这些数字
        - 如果字符串不是由纯数字组成
            - 则从第一个位置开始进行转换
            - 一直转换到非数字部分停止
            - 此时转换了多少数字,则最终结果就是这些数字
        - 如果字符串的第一个位置就已经不是数字了
            - 则无法转换
            - NaN--Not a Number
- parseFloat()
    - 将一个字符串转换为数字
    - 可以转换为整数,也可以转换为浮点数
    - 转换规则:
        - 如果当前字符串由纯数字组成
            - 则直接转换为这些数字
        - 如果字符串不是由纯数字组成
            - 则从第一个位置开始进行转换
            - 一直转换到非数字部分停止
            - 此时转换了多少数字,则最终结果就是这些数字
        - 如果字符串的第一个位置就已经不是数字了
            - 则无法转换
            - NaN--Not a Number
- charAt(index)
    - 获取指定索引位置index处的字符
    - 索引位置从0开始
- subString(index)
    - 截取字符串
    - 从指定的索引位置index处开始截取,一直截取到最后
- subString(begin,end)
    - 截取字符串
    - 从指定的索引位置begin处开始截取,一直截取到指定的索引位置end处为止(不包含end)
    - 左闭右开
- replace()
    - 将指定的字符串替换为指定的新字符串

- 可以使用正则表达式来匹配

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>字符串方法</title>
    <script>
        function f1(){
            let s = " abcD1234 aDmIn ";
            console.log("字符串长度:"+s.length);
            console.log("大写:"+s.toUpperCase());
            console.log("小写:"+s.toLowerCase());
            console.log("去除首尾空格:"+s.trim().length);
            console.log(s.charAt(5));
            console.log(s.substring(10));
            console.log(s.substring(10,15));
            // s = s.replace("a","x");
            s = s.replace(/a/g,"x");
            console.log(s);
        }
        function f2(){
            let s = "12345.67abc";
            console.log(typeof s);
            // s = parseInt(s);
            s = parseFloat(s);
            console.log(s);

        }
    </script>
</head>
<body>
<input type="button" value="字符串方法" onclick="f1()"><br/>
<input type="button" value="转换" onclick="f2()"><br/>
</body>
</html>
```

7.函数类型

7-1 函数的特点

- 使用关键字`function`定义函数
- 函数可以有返回值,但是没有返回值类型
- 函数没有参数类型
- 调用函数传递的参数可以与定义函数的参数数量不一致
- 函数的return关键字用于返回当前方法的返回值
    - 如果没有return关键字,则该方法仍然有返回值
    - 值为:`undefined`
- 函数可以作为某一个变量的值

- `var s = fn();`
  - 将fn方法最终的返回值作为变量的值赋予变量
- `var s = fn;`
  - 将整个fn方法作为变量的值赋予变量
  - 此时该变量的类型我们即认为是一个函数类型
  - 此时`s()`即表示运行该函数

7-2 函数的参数

在js中,调用函数时传递的参数并不是在定义function时所传递的形参来获取的

`function`方法的参数列表仅仅只是给传递过来的参数取了一个别名

真正获取传递的参数的值的是一个js的内置对象:`arguments`

`function`方法的参数列表仅仅只是按照参数的顺序取了一个别名

例如:`function f1(a,b)`

相当于:`a=arguments[0]`,`b=arguments[1]`

如果`functon`方法的参数数量超过了实际调用函数时传递的参数数量

则超过的部分的变量的值为:`undefined`

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>函数的特点</title>
    <script>
        function f1(a,b){
            // console.log(a+b);
            // arguments
            // for(let i = 0; i < arguments.length; i++){
            //     console.log(arguments[i]);
            // }
            console.log(a);
            console.log(b);
            // var b;
            // console.log(b);

            var s = f2();
            console.log(s);
            var s2 = f2;
            alert(s2);

            s2();
        }

        function f2(){
            console.log("用于测试");
            // return "admin";
        }
    </script>
</head>
<body>
<input type="button" value="测试" onclick="f1(1)">
</body>
</html>
```

7-3 函数的定义方式

- 定义方式一

```
function f1(){
    //此时s的结果即为fn整个函数体
    //且不会去执行fn的函数内容
    var s = fn;

    //想要执行fn方法有两种方式
    fn();
    s();
}

function fn(){
    console.log("这是一个函数");
}
```

- 定义方式二

```
function f2(){
    var s = function(){
        console.log("这是一个函数");
    };
    // alert(s);
    //想要运行该函数,通过以下方式
    s();
}
```

- 定义方式三
  - lambda表达式
  - es6引入的语法
  - 如果方法的方法体中只有return xx的代码
    - 则可以省略`function`,`{}`,`reutrn`

```
// let s = function(){
//     return "admin";
// };
//使用lambda表达式优化为以下写法
let s = () => "admin";
```

  - 如果方法的方法体中不仅仅只有return xx的代码
    - 则`{}`与`return`不可以省略

```
// let f = function(){
//     console.log("这是一个函数");
//     return "admin";
// };
//使用lambda表达式优化为以下写法
let f = () => {
    console.log("这是一个函数");
    return "admin";
};
```

8.日期类型

8-1 定义方式

- var date = new Date()
  - 没有参数
    - 获取当前时间
    - 可以通过方法改变时间的值
    - `var date = new Date();`
  - 存在参数
    - 传递的是日期的值,定义固定的某一天
    - 在日期中,月份取值范围为:`0-11`
    - 因此传递的月份的值,应该是真实需要的月份值-1
    - 当传递的值发生了变化,则具体的日期也会随之变化
    - `date = new Date(2019,1,20);`
- var date = new Date(millseconds)
  - 通过距离1970年1月1日的毫秒数来定义日期的值

8-2 常用方法

- `getFullYear()`:获取年份信息
- `getMonth()`:获取月份信息,值的范围:`0-11`
- `getDate()`:获取本月第几天
- `getHourse()`:获取当天的小时数
- `getMinutes()`:获取当前的分钟数
- `getSeconds()`:获取当前的秒数
- `getMillSeconds()`:获取当前的毫秒数
- `getDay()`:获取本周第几天,值的范围:`0-6`,一周是从周日开始的
- 以上方法均有对应的set方法
  - 表示设置对应的值

8-3 练习

8-3-1 练习一

根据给定的年份与月份求出本月一共多少天

```
function f4(year,month){
    //根据给定的年份与月份求出本月一共多少天
    //无法直接通过参数获取这一年这一个月的最后一天
    // var date = new Date(year,month-1,x);
    //但是可以取出最后一天的下一天,即下一个月的第一天
    //这一天的前面一天即为这个月的最后一天
    // var date = new Date(year,month-1+1,1);
    // var date = new Date(year,month-1+1,1-1);
    var date = new Date(year,month,0);
    console.log(year+"年"+month+"月一共有:"+date.getDate()+"天");
}
```

8-3-2 练习二

打印当前时间,要求最终的日期格式为:`yyyy年MM月dd日 HH:mm:ss`

```
function f5(){
    var date = new Date();

    console.log(date.getFullYear()+"年"+fn(date.getMonth()+1)+"月"+fn(date.getDate())+"日 "+fn(date.getHours())+":"+fn(date.getMinutes())+":"+(date.getSeconds()));
}
function fn(s){
    s = "0" + s;
    return s.substring(s.length-2);
    //2     --->        02
    //8     --->        08
    //10    --->        010
}
```

8-3-2 练习三

```
在2012年12月31日执行以下代码,求最终变量a的值为: (B)
var d = new Date();
d.setFullYear(2000)
d.setMonth(1);
var a = d.getFullYear+"-"+d.getMonth()+"-"+d.getDate();
A.2000-1-31 B.2000-2-2 C.2000-2-29 D.2000-3-2

var d = new Date(2000,1,29+2)
这一天是2000年2月31日
但是2月份只有29天
因此最终日期应该是2000年3月2日
```

9.数组类型

9-1 定义方式

- `var arr = new Array()`

    - 创建一个数组
    - 该数组的初始长度为0
    - 在js中,数组的长度是可变的
    - 当向数组中添加元素的时候,会自动扩展数组的长度
    - 当数组的元素没有赋值的时候,默认为: `undefined`
- `var arr = new Array(i)`
    - 此处的参数只能是一个数字
    - 创建一个数组
    - 该数组的初始长度为i
    - 在js中,数组的长度是可变的
    - 当向数组中添加元素的时候,会自动扩展数组的长度
    - 当数组的元素没有赋值的时候,默认为: `undefined`
- `var arr = new Array(元素1,元素2,元素3...)`
    - 此处的参数值是任意类型
    - 表示创建一个数组
    - 该数组的初始元素为这些参数的值
    - 初始长度即为这些元素的数量
    - 在js中,数组的长度是可变的
    - 当向数组中添加元素的时候,会自动扩展数组的长度
    - 当数组的元素没有赋值的时候,默认为: `undefined`
- `var arr = [元素1,元素2,元素3...]`
    - 该方式称之为json格式的定义方式
    - 用法与第三种完全一致
    - 此处的参数值是任意类型
    - 表示创建一个数组
    - 该数组的初始元素为这些参数的值
    - 初始长度即为这些元素的数量
    - 在js中,数组的长度是可变的
    - 当向数组中添加元素的时候,会自动扩展数组的长度
    - 当数组的元素没有赋值的时候,默认为: `undefined`

9-2 常用方法与属性

- length属性
    - 设置或者获取数组的长度
- sort()
    - 对于数组进行升序排序
    - 是按照字典顺序进行排序
    - 此处的比较并不是根据值来比较大小
    - 而是将其作为字符串来进行比较
    - 可以传递参数,参数是一个函数类型
        - 该函数中编写你想要的排序规则
- reverse()

    - 将数组中元素的位置颠倒过来
- join(s)
    - 将数组中的元素通过给定的参数s进行连接
    - 将其连接成一个字符串
    - 参数为可选参数
    - 如果不传递参数,默认以逗号连接
- slice(index)
    - 截取数组元素
    - 从指定的索引位置index处开始截取
    - 一直截取到最后一个元素
- slice(begin,end)
    - 截取数组元素
    - 从制定的索引位置begin处开始截取
    - 一致截取到指定的索引位置end处为止
    - 左闭右开
- toString()
    - 将数组的元素通过逗号拼接成一个字符串

10.JSON

10-1什么是JSON

`JavaScript Object Notation`

是一种轻量级的数据交换格式

采用的是与编程

语言无关的文本格式

易于编写与阅读,易于解析

10-2 语法

由一系列的键值对所组成

在定义JSON属性名的时候

建议不要使用数字开头

```
{"属性名":属性值,"属性名":属性值...}
```

10-3 使用

- 基本语法

```
var user = {"id":1,"username":"admin","age":24,"gender":"男"};
console.log("id:"+user.id);
console.log("username:"+user["username"]);
console.log("username:"+user['username']);
```

- 属性中包含对象

```
// var user = {"id":1,"name":{"firstName":"zhang","lastName":"san"}};
// console.log("firstName:"+user.name.firstName);

var name = {"firstName":"zhang","lastName":"san"};
var user = {"id":1,"name":name};
console.log(user.name);
console.log(user["name"]["lastName"]);
console.log(user.name["lastName"]);
```

- 对象集合

```
var users = [{
    "id":1,
    "username":"admin"
},{
    "id":2,
    "username":"alice"
},{
    "id":3,
    "username":"tom"
}];
console.log(users[0].id)
console.log(users[1]["username"]);
```

11.对象类型

11-1 什么是对象类型

- JavaScript中的对象类型可以理解为Java中的引用数据类型
- 对象类型包含JavaScript中的复合数据类型
  - 例如:数组、日期
- 可以自定义对象
- JavaScript中对象类型通过typeof方法返回的都是object

11-2 对象的定义方式

11-2-1 通过内置的`Object`来实现

```
var obj = new Object();
//可以定义该对象有哪些属性
//Object对象的属性可以随意写
//obj.xxx = 值相当于Java中对象的set方法
//如果直接输出obj.xxx相当于Java中对象的get方法
obj.id = 1;
obj.aa = "admin";
console.log("id:"+obj.id);
console.log("aa:"+obj.aa);
```

11-2-2 自定义对象

```
function f2(){
    var user = new User(1,"admin");
    console.log("id:"+user.id);
    console.log("name:"+user.name);
}

//此处的User函数相当于Java中的构造函数
function User(id,name){
    this.id = id;
    this.name = name;
}
```

11-2-3 JSON格式对象

```
var user = {
    "id":1,
    "name":"admin",
    "play":function(){
        console.log(this.name+"正在玩");
    }
};
console.log(typeof user);
console.log("id:"+user.id);
console.log("name:"+user.name);
user.play();
```

11-2-4 通过JSON格式的字符串来进行操作

```
var user = "{'id':1,'name':'admin'}";
console.log(typeof user);
//将json格式的字符串转换为json对象
//方法一:通过eval方法来实现
var obj = eval("("+user+")");
console.log(typeof obj);
console.log(obj.id);
console.log(obj.name);

//方法二:通过JSON.parse方法进行转换
//该方式只能使用单引号作为字符串的标识
//该方式安全性较高
user = '{"id":1,"name":"admin"}';
obj = JSON.parse(user);
console.log(obj);
```