1. JavaScript 鸟瞰

JavaScript 实际上是 ECMAScript 语言规范的一个实现

- types
- properties
- values
- function
- reserved words

ECMAScript

JavaScript

Web浏览器：BOM / DOM 等
Web服务器：请求 / 文件 / 共享数据 等

运行环境

- **浏览器 (jQuery / Angular / React)**
- 服务器 (Node.js / io.js / Grunt / Gulp / Babel)
- App (React Native)
- Desktop (Electron)

在浏览器中编程

- 非侵入式JavaScript是一种将Javascript从HTML结构抽离的设计概念，避免在HTML标签中夹杂一堆onchange、onclick等属性去**挂载Javascript事件**，让HTML与Javascript分离，**依模型-视图-控制器**的原则将功能权责清楚区分，使HTML也变得结构化容易阅读。

基本原则

- 行为层和表现层分离开；
- 是解决传统JavaScript编程问题（浏览器呈现不一致，缺乏扩展性）的最佳实践；
- 为可能不支持JavaScript高级特性的用户代理（通常是浏览器）提供渐进增强的支持。

如何做?

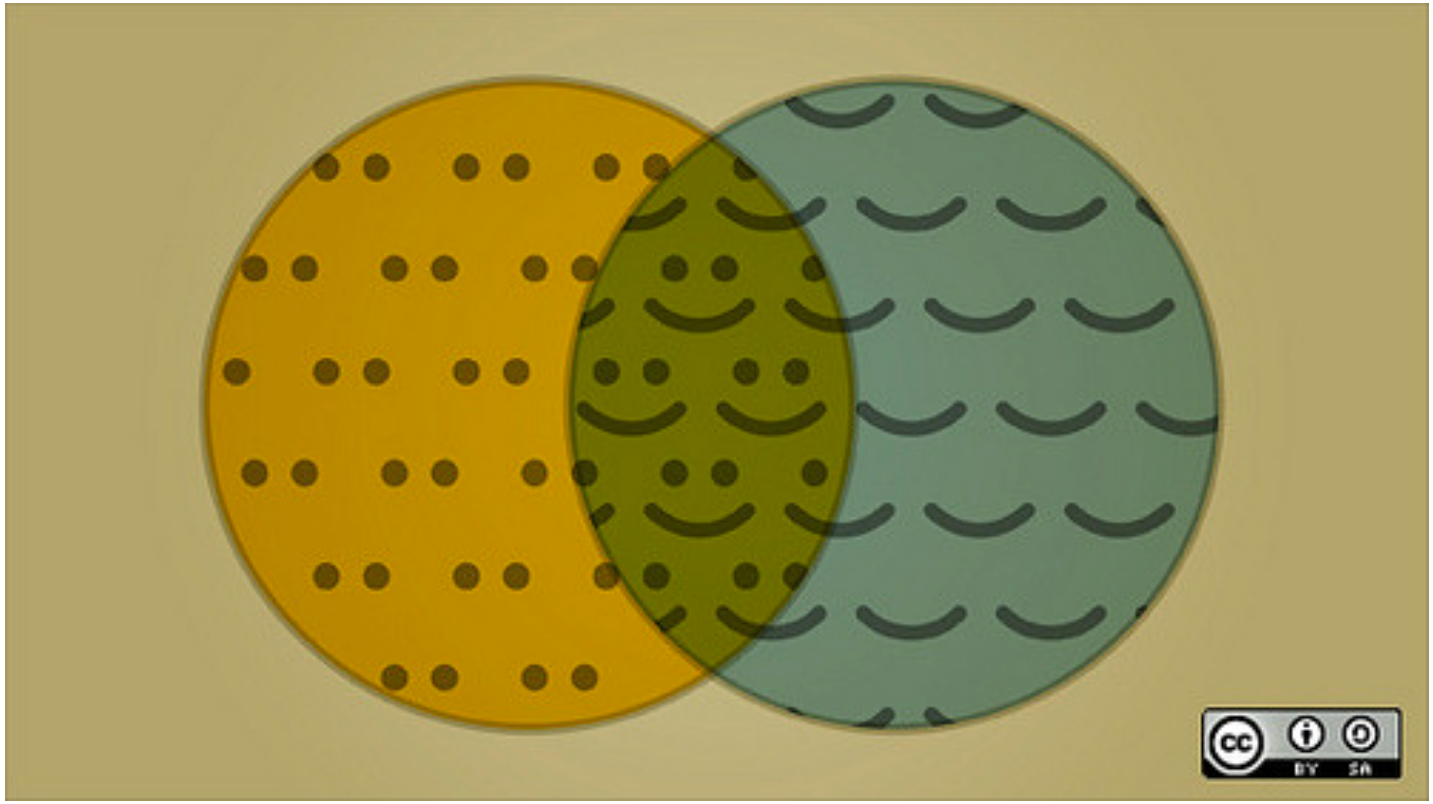
Before

```
<input type="text" name="date" onchange="validateDate()" />
```

After

```
<!-- HTML: -->
<input type="text" name="date" id="date" />

<!-- JavaScript: -->
window.onload = function() {
    document.getElementById('date').onchange = validateDate;
};
```



CSS & JavaScript

JavaScript 代码库

- JavaScript 代码块 {}
- 代码块可以作为函数表达式的主要部分
  var foo = function (){}
- 代码块可以作为函数声明的主要部分
  funcation bar(){}
- 代码块可以作为控制流的一部分 if(){}

只有函数才产生新的作用域

```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2;  // same variable!
    console.log(x);  // 2
  }
  console.log(x);  // 2
}
```

```
function varTest() {
  if (true) {
    x = 2;  // same variable!
  }
  console.log(x);  // 2
}
```

（除了let和const，它们是block-scoped）

立即调用函数表达式

```
// 推荐
(function() {
    console.info( this );
    console.info( arguments );
}( window ) );


// 推荐
(function() {
    console.info( this );
    console.info( arguments );
})( window );

// 推荐
;!function() {
    console.info( this );
    console.info( arguments );
}( window );
```

```
// 不推荐
!function() {
    console.info( this );
    console.info( arguments );
}( window );

// 不推荐
+function() {
    console.info( this );
    console.info( arguments );
}( window );
```

IIFE（Immediately Invoked Function Expression）

- 它提供了一个闭包来防止命名冲突
- 它提供了优雅的块级作用域
- 它可以防止污染全局命名空间
- 它促进了代码的模块化

JavaScript 总是按值传递参数

- It's always pass by value, but for objects the value of the variable is a reference. Because of this, when you pass an object and change its members, those changes persist outside of the function. **This makes it look like pass by reference.** But if you actually change the value of the object variable you will see that the change does not persist, proving it's really pass by value.

```
function changeObject(x) {
  x = {member:"bar"};
  alert("in changeObject: " + x.member);
}

function changeMember(x) {
  x.member = "bar";
  alert("in changeMember: " + x.member);
}

var x = {member:"foo"};

alert("before changeObject: " + x.member);
changeObject(x);
alert("after changeObject: " + x.member);
alert("before changeMember: " + x.member);
changeMember(x);
alert("after changeMember: " + x.member);
```

函数最佳实践

- 一般认为函数是黑盒，假定函数返回变量时只会影响封闭的作用域，**不会影响传入的参数**。

函数声明vs函数表达式

- JS是按照代码块来进行编译和执行的，代码块间相互独立，但变量和方法共享。
- 在JS的预编译期，声明式函数将会先被提取出来，然后才按顺序执行js代码。
- 分别对每一个代码块进行预编译和执行，语法错误直接跳到下一个代码块。

- 函数声明

```
function sayHi(){
 alert(‘Hi!’);
}
```

- 函数表达式

```
var sayHi = function(){
 alert(‘Hi!’);
}
```

函数声明提升

- 函数声明提升，是指在执行代码之前，会先读取函数声明

```
sayHi(); // Hi!
function sayHi(){
  alert('Hi!');
}
```

- 函数表达式与其他表达式一样，在使用前必须赋值

```
sayHi(); //错误！函数还不存在
var sayHi = function(){
  alert('Hi!');
}
```

如何在条件控制器中声明函数

```
if(true){
  function sayHi(){
    alert('Hi!');
  }
}else {
  function sayHi(){
    alert('Yo!');
  }
}
sayHi();//不确定，根据解释器不同而不同
```

```
if(true){
  var sayHi = function(){
    alert('Hi!');
  }
}else {
  var sayHi = function(){
    alert('Yo!');
  }
}
sayHi();//Hi!
```