信息工程系

教

案

课程名称：JavaScript 程序设计

教　　师：黄苗苗

总 学 时 ：108

理论学时：36

实训学时：72

上课班级：计算机 251、计算机 3+ 251

授课学期：25-26（2）

课 程 性 质

JavaScript 是一种基于对象和时间驱动并具有安全性能的脚本语言。使用它，可以与 HTML 超文本标记语言、Java 语言脚本语言一起实现在一个 web 页面中链接多个对象，与 web 客户交互的作用。从而开发客户端的应用程序。掌握 JavaScript 编程技术是当代计算机科学与技术、网络应用开发等专业学生必修的专业课程。

课程要求侧重掌握为将来从事网页开发、手机应用开发、网络程序应用开发等提供必要的基础知识，打下良好的基础。因此要求上述专业学生都必须掌握本课程的内容。

课程要求：

1. 了解 JavaScript 的发展历史
2. 掌握 JavaScript 的基本语法
2. 应用 JavaScript 进行简单网页互动开发

课程性质：

计算机基础专业课程，本课程是对 JavaScript 做导入的介绍，并介绍基本的网页开发程序应用背景知识，关系以后网络技术编程的基础、是当前网络应用开发的前景知识准备，需要学生对此投入较大的学习量，并将其应用到实际操作中才能切实掌握该技术。

课程教学手段

依据课程性质及周课时安排：

理论课（2 节/周）：

课程主要是从总体概述之后，首先导入 JavaScript 的基础语法，最后结合应用，介绍如何使用 JavaScript 高级设计。

结合课程的各层结构，在每个章节的授课中，分几个主要方式进行：

- 应用环境导入：结合知识点，导入可能见到的情境，对情景进行剖析，结合学生掌握点，分析已经解决问题及函待处理问题，由此导入新知识点。
- 承上启下式进入：对于上一个章节，进行总结，然后在上一章节的基础上，对新的可能出现的情境进行引导式引入，然后就新的问题的本质、现象出现的方式以及其如何工作引入思考，然后进入章节介绍。
- **课程思政体现**：教学主要体现在以学生为中心。比如，各环节问题引入，结合了抽调个人学生问题——比如实验遇见问题，及时对应处理。同时以班级学生特点，围绕专业发展方向，及时调整，辅助学生完善专业技能学习发展。

引入章节之后，对于每个技术要点，依据各个特点进行逐一详细解析，帮助学习者尽快进入新知识的学习。主要各要点归纳如下：

- 本技术功能概述
- 技术解决问题情境
- JavaScript 相关知识点
- JavaScript 逐个技术语法解析
- JavaScript 该技术各要点集合汇总
- JavaScript 该技术综合实例设计剖析

实验课（2 节/周）：

针对本课程特性，实操性实验课是检验学生掌握程度和加深学生技能的重要过程。

实际结合网络知识以及日常工作可能布置网络应用设计情况设计。需要从浅入深，有基础语法的小实验一步一步扎实练习，最后才能汇总为大的设计完成。

实验课的难度系数属于中等难度系数，需学生课后配合多加练习。

第一章 JavaScript概 述

❄ 计划课时：

4 学时

❄ 教学目的：

1. 了解 JavaScript 的发展历史
2. JavaScript 模型
3. 网页中嵌入 JavaScript
4. 执行 JavaScript 程序
5. 浏览器与 JavaScript
6. JavaScript 开发工具
7. 错误处理

❄ 教学重点、难点：

1. 掌握：JavaScript 模型。
2. 熟悉：JavaScript 开发工具。
3. 了解：JavaScript 未来发发展趋势。

❄ 教学内容：

JavaScript 是面向 Web 的编程语言；获得几乎所有的网页浏览器的支持；网页设计&Web 应用必须掌握的基本工具

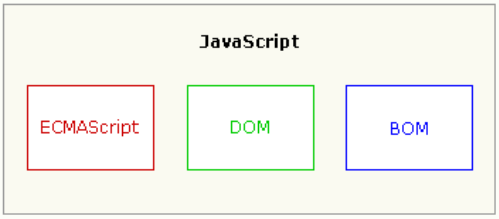
❄ 发展历史

- ♪ 1997，欧洲计算机制造商协会 ECMA，ECMAScript
- ♪ 1998，浏览器开发商开始致力于将 ECMAScript 作为各自 JavaScript 实现的参考标准
- ♪ ECMAScript 是 JavaScript 语言的国际标准， JavaScript 是 ECMAScript 的一种实现

- 2015 年， ECMAScript 6 发布正式版本，且，更名为 ECMAScript 2015
  - 这是继 ECMAScript 5 之后一次重大改进
  - 增加了许多必要的特性
    - 模块、类、实用特性等

JavaScript 的组成部分

- 核心：ECMAScript
- 文档对象模型：DOM
- 浏览器对象模型：BOM



- 核心：ECMAScript
  - WEB 是 ECMAScript 实现的宿主环境之一
    - 宿主环境：提供基本的 ECMAScript 实现，也提供各种功能扩展
- 文档对象模型：DOM
  - Document Object Model
  - 针对 XML 但经过扩展用于 HTML 的应用程序编程接口 API
  - DOM 把整个页面映射为一个多层节点结构
    - HTML 或 XML 页面中的每个组成部分都是某种类型的节点
    - 这些节点包含着不同的数据类型
  - DOM1→→ DOM2 →→DOM3
- 浏览器对象模型：BOM
  - 可以对浏览器窗口进行访问和操作
  - 只是 JavaScript 的一个部分，没有任何相关的标准
  - 主要处理浏览器窗口和框架

在网页中嵌入 JavaScript 脚本

- 在 HTML 页面中，如果要嵌入 JavaScript 脚本，需要加入<script>标签
- 使用<script>标签有两种方式：
  - 直接在页面嵌入 JavaScript 代码
  - 包含外部 JavaScript 文件（扩展名为 .js）
- 直接在页面嵌入 JavaScript 代码
  - 新建 HTML 文档
  - 在<head>标签中插入<script>标签

- ✧ 为<script>标签指定 type 值为“text/javascipt”
- ✧ 在<script>标签内部输入 JavaScript 代码
- ✧ 保存文档
- ✧ 浏览器浏览

♪ 包含外部 JavaScript 文件（扩展名为 .js）

- ✧ 新建.js 文本文件
- ✧ 在.js 文本文件中编写 JavaScript 代码
- ✧ 保存.js 文本文件文件——与调用的网页文件于同一目录之下
- ✧ 新建 HTML 文档
- ✧ 在<head>标签中插入<script>标签
- ✧ 为<script>标签指定 type 值为 “text/javascipt”，src 指向外部.js 文本文件的路径
- ✧ 调用 JavaScript 代码
- ✧ 保存文档
- ✧ 浏览器浏览

♪ 脚本位置：

- ✧ 所有<script>标签都会按照它们在 HTML 中出现的顺序依次被解析
- ✧ 一般只有解析完前面的<script>标签中的代码之后，才会开始解析后面<script>标签中的代码
- ✧ 默认情况下：
  - ➢ HTML 中<script>标签都放在页面头部<head>标签中
  - ➢ 缺点：
    - ■ 必须等到全部的 JavaScript 代码都被下载、解析和执行完后，才开始呈现页面内容
- ✧ WEB 中：
  - ➢ 一般会把<script>标签引用放在<body>标签中页面的内容后

❄ 执行 JavaScript 程序

♪ JavaScript 的解析过程

- ✧ 预处理/预编译
  - ➢ JavaScript 解释器将完成对 JavaScript 代码的预处理操作——把 JavaScript 代码转换成字节码
- ✧ 执行
  - ➢ JavaScript 解释器把字节码生成二进制机械码，并按顺序执行

♪ 预处理/预编译

- ✧ 词法分析：对 JavaScript 脚本逐一分析是否符合 JavaScript 规范，是否与语法错误
- ✧ 语法分析：把从程序中手机的信息存储到数据结构中
  - ➢ 符号表：存储程序中所有符号的一个表;包括字符串、直接量、变量名、函数名等
  - ➢ 语法树：构建程序结构的一个树形表示，并将使用这个树形结构来生成中间代码

- 执行过程：
  - HTML 文档在浏览器中的解析过程（ JavaScript 同）：
    - 按 文件流 从上到下 逐步解析 页面结构 和 信息
  - JavaScript 解释器在执行脚本时，是按 块 来执行的

- 响应事件：
  - JavaScript 的响应操作是通过事件驱动的模式实现的
  - 特点：
    - 事件发生具有不确定性
    - 响应顺序也有不确定性

- 动态脚本:
  - 设计页面显示的内容是包含<script>标签的代码字符串序列则
    - 页面首先解析显示页面内容
    - 遇见<script>标签，则在解析相对应代码块并执行

    如：
    - document.write()，
    - 先把输出的字符串写入到标签所在的文档的位置，
    - 浏览器解析完 document.write()所在的文档内容后，
    - 继续解析 document.write()输出的内容，
    - 然后才按顺序解析后面的 HTML 文档

❄ 代码测试&错误处理

- 主要内容：
  - 浏览器设置
  - 开发工具
  - 错误处理

- JavaScript 寄生于 Web 浏览器中
  - 目前主流的浏览器有：
    - IE
    - Firefox
    - Opera
    - Safari
    - Chrome

- 浏览器**设置**：
  - 内核：
  - 渲染引擎：
    - 取得网页内容、整理信息、计算网页的显示方式，输出
  - JavaScript 引擎：
    - 解析 JavaScript 脚本，执行 JavaScript 代码实现网页的动态效果
  - 错误报告：

- 浏览器都具有某种 JavaScript 的错误报告机制，默认情况下都会隐藏此类消息
- 用户需要启用浏览器的 JavaScript 报告功能以收集错误信息

✧ IE 设置：

♪ 开发工具：

✧ 常见编辑器：

- 文本编辑器、Dreamweaver 等

✧ 测试调试平台：

- IE 等都提供了 JavaScript 控制台——查看 JavaScript 错误、并允许通过 diamante 想控制台输出消息

✧ console 对象方法：

- error(message)
- info (message)
- log (message)
- warm (message)

♪ 错误处理

✧ JavaScript 是松散类型的语言：

- 类型转换错误
- 数据转换错误
- 通信错误

✧ 7 种错误类型：

- Error
- EvalError
- SyntaxError
- RangeError
- ReferenceError
- TypeError
- URLError

✧ 错误处理

- try-catch 语句
  - catch 块会接收到一个包含错误信息的对象
  - 适合处理那些无法控制的错误
- finally 子句
  - 如果已经被使用，则其代码无论如何都会执行!!!
  - 如果提供了 finally 子句，则 catch 子句就成了可选的
- throw 操作符
  - 用于随时抛出自定义错误
  - 抛出错误时，必须要给 throw 指定一个值——可以是任何类型
  - IE 只有在抛出 Error 对象的时候才会显示自定义错误信息
  - 抛出错误的目的
  - 提供错误发生具体原因的信息
- error 事件

- 任何没有通过 try-catch 处理的错误都会出发 windows 的 error 时间

第二章 JavaScript核心编程

❄ 计划课时：

18 学时

❄ 教学目的：

1. JavaScript 基本语法
2. 运算符
3. 程序结构
4. 数组
5. 函数
6. 对象

❄ 教学重点、难点：

1. 掌握：
   a) JavaScript 的变量定义与使用；
   b) 如何在 JavaScript 中合理利用运算符和程序结构；
   c) 高效使用数组和对象；
   d) 定义、调用函数。
2. 熟悉：数组、函数以及对象。
3. 了解：闭包函数如何灵活适应需求。

❄ 教学内容：

本部分涵盖内容比较多，主要有：

- **基本语法**
- **运算符**
- **程序结构**
- **数组**

- 函数
- 对象

分述如下：

基本语法

基本词法

变量

数据类型

严格模式

案例分析

基本语法——基本词法

- JavaScript 词法
  - 字符编码
  - 命名规则
  - 标识符
  - 关键字
  - 注释规则
  - 特殊字符用法
- 字符编码
  - JavaScript 程序使用 Unicode 字符集编写
  - Unicode 字符集中每个字符使用 2 个字节来表示
    - 故此可以使用 中文 ——极其不推荐
  - ！ JavaScript 一般寄存在网页中， 须由浏览器来计时，
  - 故此，
    - JavaScript 的编程过程中，
    - 最好考虑进嵌入页面的字体
- 区分大小写
  - JavaScript 严格区分大小写
  - 一般情况下， JavaScript 都采取小写格式
  - 例外：
    - 定义 JavaScript 构造函数时，让函数名首字母大写
    - 多次组成的变量名，考虑部分大写
- 标识符：
  - JavaScript 中的标识符主要包括：
    - 变量名 函数名 参数名 属性名

- 规则：
  - 第一个字符必须是字母、下划线或美元符号
  - 其他可以使用字母、数字、下划线、美元符号
  - 不能与 JavaScript 关键字或保留字同名
  - 可以使用 Unicode 中的转义字符
- Eg:
  - var \u0061bc = "标识符 abc(变量）中 a 字符的 Unicode 转义序列是\\u0061";
  - alert(\u0061bc);

♪ 关键字&保留字

- ECMA-262 定义了 ECMAScript 支持的一套关键字（keyword)。
- 这些关键字标识了 ECMAScript 语句的开头和/或结尾。
- 根据规定，关键字是保留的，不能用作变量名或函数名。
- ECMA-262 定义了 ECMAScript 支持的一套**保留字（reserved word)**。
- 保留字在某种意思上是为将来的关键字而保留的单词。因此保留字不能被用作变量名或函数名
-

break	delete	function	return	typeof
case	do	if	switch	var
catch	else	in	this	void
continue	finally	instanceof	throw	while
default	for	new	try	with

♪ 分隔符：

- 用来分隔代码中的各种几号
  - 空格、制表符、换行符、换页符
- 解析时，会忽略分隔符

♪ 注释：

- //
- /* */

♪ 转义序列：

- 以\u 为前缀，跟上 4 个 16 进制数

❄ 基本语法——变量

♪ 声明：

- 变量是用 var 运算符（variable 的缩写）加变量名定义
- 可以用一个 var 语句定义两个或多个变量,逗号分隔
- 注意
  - 可以不声明类型
  - 是在使用变量之前不是必声明
  - 使用 var 声明的变量是永久性的，无法使用 delete 运算符删除

♪ 赋值：

- 声明：预处理

- 赋值：执行期

♪ 作用域：
- 变量在程序中可访问的有效范围
- 分类：
  - 全局作用域
    定义方法：
    - 在任何函数外面直接执行 var 语句
    - 直接添加一个属性到全局对象上
    - 在 web 中，全局对象是 window
    - 直接使用未经声明的变量
  - 局部作用域

♪ 避免变量污染：
- 把多个全局变量都追加到一个名称空间，可以降低与其他程序产生冲突的几率
- 使用函数体将信息隐藏

❄ 基本语法——数据类型

♪ JavaScript 是弱类型语言
- JavaScript 中的 6 种基本数据类型
  - null
  - undefined
  - number
  - string
  - Boolean
  - object

♪ 检测--**typeof 运算符**
- typeof 运算符有一个参数，即要检查的变量或值。
- 对变量或值调用 typeof 运算符将返回下列值之一：
  - undefined - 如果变量是 Undefined 类型的
  - boolean - 如果变量是 Boolean 类型的
  - number - 如果变量是 Number 类型的
  - string - 如果变量是 String 类型的
  - object - 如果变量是一种引用类型或 Null 类型的

♪ 数值：
- Number 类型是最特殊的类型：
  - 这种类型既可以表示 32 位的整数，还可以表示 64 位的浮点数。
- 直接输入的（而不是从另一个变量访问的）任何数字——数值直接量，都被看做 Number 类型的字面量
- 整数可被表示为八进制（以 8 为底）或十六进制（以 16 为底）的字面量——参与数学运算后为十进制格式。

- 八进制字面量的首数字必须是 0,其后的数字可以是任何八进制数字（0-7）
- 十六进制的字面量，首位数字必须为 0，后面接字母 x，然后是任意的十六进制数字（0 到 9 和 A 到 F）。这些字母可以是大写的，也可以是小写的

✧ 数学运算：
- Math 对象的属性或方法
- toString()方法

✧ 特殊数值：
- **Infinity**
- **NaN**
- **Number.MAX_VALUE**
- **Number.MIN_VALUE**
- **Number.NaN**
- **Number.POSITIVE_INFINITY**
- **Number.NEGATIVE_INFINITY**

✧ NaN，表示非数（Not a Number）。
- NaN 是个奇怪的特殊值。
- 一般说来，这种情况发生在类型（String、Boolean 等）转换失败时。
- 例如：
- 要把单词 blue 转换成数值就会失败，因为没有与之等价的数值。
- 与无穷大一样，NaN 也不能用于算术计算。
- NaN 的另一个奇特之处在于，它与自身不相等

✧ 推荐
- 使用 isNaN（）

♪ 字符串：

✧ 无区分字符串和字符
- 它是唯一没有固定大小的原始类型。

✧ 可以用字符串存储 0 或更多的 Unicode 字符

✧ 字符串由 unicode 字符、数字和各种符号组成，必须包含在单引号或双引号之中

✧ 字符串中每个字符都有特定的位置，首字符从位置 0 开始，第二个字符在位置 1，依此类推

✧ 引号需要使用转义字符

✧ 可以使用加号（+）运算连接两个字符串

✧ length 属性可以确定字符串长度

♪ 布尔值

✧ 在 JavaScript 中:

undefined	null	“”	0	NaN	false

- 这 6 个值准环卫逻辑值时为 false
- 其余任何类型的数据转换时都为 true

♪ Null

- 表示空值

♪ Undefined
- 表示未定义的值

❄ 基本语法——严格模式

♪ 严格模式：
- 就是使 JavaScript 在更严格的条件下运行

♪ 优势：
- 消除 JavaScript 语法的一些不合理、不严谨之处
- 消除一些代码运行的不安全之处
- 提高编译器效率
- 为未来准备

♪ 启动严格模式：
- 在代码首部（前面没有任何有效的 JavaScript 代码）加入——注释字符串：
- use strict

♪ 严格模式的应用场景
- 全局模式
  - “use strict”放在脚本的第一行
  - 整个脚本都将以严格模式运行
- 局部模式
  - use strict”放在函数内第一行
- 严格模式应用场景——**模块模式**
  - 借用局部模式方法，将真个脚本文件放在一个立即执行的匿名函数中
- 严格模式——**执行限制**
  - 显示声明变量
  - 静态捆绑
  - 增强的安全措施
  - 禁止删除变量
  - 显示报错
  - 重名错误
  - 禁止八进制表示法
  - arguments 对象的限制
  - 函数必须声明在顶层
  - 保留字
  - 动态捆绑

❄ 基本语法——案例分析

♪ 类型检测：
- typeof 运算符

- ✧ Object 对象的 constructor 属性
- ✧ 封装类型检测方法：toString()
- ✧

```
// 安全检测JavaScript基本数据类型和内置对象
// 参数：o表示检测的值
// 返回值：返回字符串"undefined"、"number"、"boolean"、"string"、"function"、
//"regexp"、"array"、"date"、"error"、"object"或"null"
function typeOf(o){
   var _toString = Object.prototype.toString;
   // 获取对象的toString()方法引用
   // 列举基本数据类型和内置对象类型，你还可以进一步补充该数组的检测数据类型范围
   var _type ={
      "undefined" : "undefined",
      "number" : "number",
      "boolean" : "boolean",
      "string" : "string",
      "[object Function]" : "function",
      "[object RegExp]" : "regexp",
      "[object Array]" : "array",
      "[object Date]" : "date",
      "[object Error]" : "error"
   }
   return _type[typeof o] || _type[_toString.call(o)] || (o ? "object" :
"null");
   // 通过把值转换为字符串，然后匹配返回字符串中是否包含特定字符进行检测
```

♪ 数字类型转换：

- ✧ 数字→字符串:+ 以及 toString 方法
- ✧ 数字→指定进制字符串：toString（参数）
- ✧ 设置转换数值的小数点格式：toFixed() toExponential() toPrecision()
- ✧ 非数字原始值→数字：parseInt() parseFloat()
  - ➢ 只有对 String 类型调用这些方法，它们才能正确运行；对其他类型返回的都是 NaN
  - ➢ 从位置 0 开始查看每个字符，直到找到第一个非有效的字符为止，然后把该字符之前的字符串转换成 shuzi
  - ➢ 使用乘号*运算符,JavaScript 能自动把数字字符串转换为数值
- ✧

```
//（1）如果采用默认模式，则toString()方法
var a = 1.000;          会直接把数值转换为数字字符串。
var b = 0.0001;
var c = 1e-4;
alert(a.toString());    // 返回字符串"1"
alert(b.toString());      // 返回字符串"0.0001"
alert(c.toString());    // 返回字符串"0.0001"

var a = 010;            // 八进制数值10
var b = 0x10;             // 十六进制数值10
alert(a.toString());    // 返回字符串"8"
alert(b.toString());      // 返回字符串"16"

//（2）如果设置参数，则toString()方法会根据参数把数值转换为对应进制
var a = 10;             // 十进制数值10
alert(a.toString(2));     // 返回二进制数字字符串"1010"
alert(a.toString(8));     // 返回八进制数字字符串"12"
alert(a.toString(16));    // 返回二进制数字字符串"a"
```

♪ 对象转换：

- ✧ →布尔值：
  - ➢ 逻辑非！运算、Boolean()构造函数
- ✧ →对象：

- 包含 String Number Function Boolean 四中基本对象构造器
- ✧ 对象→值：
  - 非空对象→true、对象→参与数值运算的数字、数组→参与数值运算的数字、
- ✧ 强制转换：
  - Boolean(value) - 把给定的值转换成 Boolean 型；
  - Number(value) - 把给定的值转换成数字（可以是整数或浮点数）；
  - String(value) - 把给定的值转换成字符串；

运 算 符

概 述

算术运算符

逻辑运算符

关系运算符

赋值运算符

对象操作运算符

案例分析

❄ 运算符 · 概 述

♪ JavaScript 定义了 51 个运算符，分为 15 个优先等级

- ✧ P65
- ✧ 一元运算符、三元运算符、赋值运算符都是从右到左执行
- ✧ 运算符只能操作特定类型的数据，返回值也是特定类型的值
- ✧ 对于能够改变运算数的运算符要注意逻辑使用，eg:=运算

❄ 运算符 · 算术运算符

♪ 算术运算符：

- ✧ ++ --
  - 运算执行顺序
- ✧ -（数值取反运算符）
- ✧ * / %
  - /：0 为除数
  - %：可适用浮点数
- ✧ + -
  - +：注意字符串
  - -：注意布尔值、它可快速把值转换为数字

- 实例分析

❄ 运算符 · 逻辑运算符

- 逻辑运算符：
  - ！
    - 返回值必定是布尔值
  - &&
    - 运算数可以是任意数据类型，返回值未必是布尔值
    - null、NaN，返回值总是 null、NaN
    - undefined 返回错误
  - ||
- 实例分析：

❄ 运算符 · 关系运算符

- 关系运算符：
  - 大小比较**< <= > >=**
    - 运算数可以是任何类型数据
    - 数字、字符串比较
    - NaN 参与比较，返回 false
  - 包含检测 **in instanceof**
    - in：判断成员 “字符串” **in** 对象/数组
    - instanceof：判断实例 对象实例名 **instanceof** 类名/构造函数名
  - 等值检测**=== !== == !=**
    - 优先转换比较：数字 字符串 引用地址
    - NaN 与任何值都不等
    - null undefined 等值 不等类型
  - 引用类型比较主要比较引用的地址
- 实例分析：

❄ 运算符 · 赋值运算符

- 赋值运算符：
  - 简单的赋值运算符由等号（=）实现，只是把等号右边的值赋予等号左边的变量。
  - 复合赋值运算是由乘性运算符、加性运算符或位移运算符加等号（=）实现的
    - 乘法/赋值（*=）
    - 除法/赋值（/=）
    - 取模/赋值（%=）
    - 加法/赋值（+=）
    - 减法/赋值（-=）

- 左移/赋值（<<=）
- 有符号右移/赋值（>>=）
- 无符号右移/赋值（>>>=）

♪ 实例分析：

❄ 运算符 · 对象操作运算符

♪ 对象操作运算符：

- ✧ in instanceof new delete . [] ()
- ✧ 操作对象：对象、数组、函数
- ✧ new
- ✧ 创建并初始化一个对象
  - ①自定义类只能通过 new 进行实例化
- ✧ delete
  - ➢ 删除指定对象的属性、数组元素或变量；返回：true 或 false
  - ①var 语句声明的变量不允许删除；
  - ②不存在的对象成员等删除会返回 true；
  - ③引用类型值删除引用不影响原值；
  - ④delete 会彻底清除内存空间
- ✧ . []
  - ①对象属性名字符串可适用[]；
  - ②[对象]会先调用 toString()转换，否则调用其 valueOf()方法；
  - ③[]布尔值转化成字符串“true”“false”

♪ 实例分析：

程序结构

语 句

分支结构

循环结构

结构跳转

案例分析

❄ 程序结构——语句

♪ 分类：

- ✧ 单句
- ✧ 复句
- ✧ P95
- ✧ 表 6.1

- 常见语句：
- 表达式语句；复合语句；声明语句；空语句

❄ 程序结构——分支结构、循环结构、结构跳转

♪ P106

♪ if **vs** switch
- 多条件 vs 单条件

♪ 常见循环结构：
- while
- do/while
- for
- for/in

♪ for/in
- *in* 后是一个对象或数组类型的表达式
  - 对于数组，property 存储的是数组元素的下标
  - 对于对象，property 存储的是对象的属性名或方法名
- for in 能枚举对象的所有成员
  - 如果对象的成员被设置为**只读、存档、不可枚举**等属性，则它无法枚举
  - 所有内置方法都不允许枚举

♪ 结构跳转主要包括：
- label
- break
- continue
- label:
- 语法： **label:statements**
  - 标签与变量名属于不同的语法体系
  - 标签与属性名属于相同的语法体系
- break：
  - break 仅限于跳出当前结构
  - break 后的 label，指示程序终止执行之后跳转到该 label 语句末尾的位置为起点继续执行
- continue：
  - 停止当前循环，执行下一次循环　P121

数组

定义数组

使用数组

数组对象

案例分析

❄ 数组******定义数组

- 有 2 种方法可以定义数组：
  - 使用构造函数创建数组：
    - new Array()
      - Array（）如果只有一个数值参数：
      - 表示定义了该数组的长度
  - 使用直接量定义数组：
    - [, , ,]

❄ 数组******使用数组

主要内容有：

- 存取数组元素
  - 下标法
  - 下标为其他值，JS 会自动转化为一个字符串，生成关联数组，作为其对象属性名字
- 数组长度
  - JavaScript 在初始化数组时，只有那些真正存储在数组中的元素才能够被分配到内存中
  - length 属性值不是数组元素的实际个数，而是包含的元素个数
  - 示例：
- 对象与数组
  - 对象：包含已命名的值的集合类型
  - 数组：包含已编码的值的集合类型
  - 关联数组：将值与特定字符串关联在一起
- 定义多维数组
  - JavaScript 不支持多维数组
  - 使用嵌套数组作为数组元素来实现多维数组

❄ 数组******数组对象

- JavaScript 为 Array 定义了 9 个原型方法：
  - 检索数组：for/in
    - 借助 length 属性，循环遍历数组
  - 操作元素：push() pop() unshift() shift() concat()
    - delete 操作既能够删除元素的值，而不是元素
    - push()
      - 给数组添加元素

- pop()
  - 删除并返回数组的最后一个元素
- unshift()
  - 在数组头部插入一个元素
- shift()
  - 将元素移出数组
- concat()——**返回一个新的数组**
  - 连接数组
- push() pop()
  - 都是在尾部执行操作
- pop()
  - 如果数组为空，则不执行任何操作，返回 undefined
- unshift() shift()
  - 在数组头部执行
- unshift()可以包含多个参数，执行一次性插入
  - shift()如果数组为空，则不执行任何操作，返回 undefined
- concat()
  - 可以跟随多个参数
  - 如果参数是数组，打散之后作为单独的元素连接，但不递归打散
  - 返回一个新数组

✧ 操作子数组：splice() slice()

- splice(起始下标,删除元素的个数,插入元素列表)
  - 起始下标：
    - 从 0 开始
    - 如果只有这个参数,则删除包含该下标的元素及其后所有元素
    - 如果为负值，则由右向左，执行倒数定位
  - 删除元素的个数：
    - 如果不删除，必须指定为 0
  - 插入元素列表：
    - 不定参数
- slice(起始下标, 结束下标)
  - [起始下标, 结束下标)
  - 参数如果为负值，则由右向左，执行倒数定位

✧ 数组排序：reverse() sort()

- reverse()
  - 颠倒数组中元素的顺序
- sort()
  - 无参数：
  - 按字母顺序对数组中的元素排序
  - 排序时，逐位比较
- undefined 元素被排在数组的末尾
  - 有参数：

✧ 排序函数：sort(args)

- 有参数：
  - 该参数是函数参数
  - 函数要比较两个值
  - 返回一个用于说明这两个值的相对顺序的数字
  - <0、 =0、 >0

✧ 数组与字符串转换：toString() toLocalString() join()

- toString()是 Object 类定义的
  - 所有对象都继承了这个方法
- toLocalString()
  - 将数字转换为本地字符串，如中国浮点数值
- join()
  - 以其分隔符参数作为连接符连接数字元素成为一个字符串

✧ 定位：indexOf() lastIndexOf()

- indexOf(searchElement[,fromIndex])
  - 返回数值在数组中的第一个匹配项
- lastIndexOf()
  - 返回数值在数组中的最后一个匹配项

✧ 迭代：forEach() every() some() map() filter()

- forEach()：为数组的每个元素调用定义的回调函数
  - array.forEach(callbackfn[,thisArg])
    - callbackfn：必须参数，最多可以接受 3 个参数的函数
    - value:数组元素的值
    - index 数组元素的数字索引
    - array:包含该元素的数组对象
    - thisArg：可选参数，callbackfn 函数中的 this 关键字可引用的对象
  - 不会为数组中缺少的元素调用回调函数
  - EG:
- every()：检查定义的回调函数是否为数值中的所有元素返回 true
  - array.every(callbackfn[,thisArg])
    - callbackfn：必须参数，最多可以接受 3 个参数的函数
    - value:数组元素的值
    - index 数组元素的数字索引
    - array:包含该元素的数组对象
    - 每个元素都会调用 callbackfn，直至返回 false 或到达数组尾部
    - thisArg：可选参数，callbackfn 函数中的 this 关键字可引用的对象
  - 不会为数组中缺少的元素调用回调函数
- some()：检查定义的回调函数是否为数值中的任何元素返回 true
  - array.some(callbackfn[,thisArg])
    - callbackfn：必须参数，最多可以接受 3 个参数的函数
      - value:数组元素的值
      - index 数组元素的数字索引

- array:包含该元素的数组对象
- 每个元素都会调用 callbackfn，直至返回 *true* 或到达数组尾部
- thisArg：可选参数，callbackfn 函数中的 this 关键字可引用的对象

- 不会为数组中缺少的元素调用回调函数

- map()：为数组的每个元素调用定义的回调函数，并返回包含结果数组
  - array.map(callbackfn[,thisArg])
  - callbackfn：必须参数，最多可以接受 3 个参数的函数
    - value:数组元素的值
    - index 数组元素的数字索引
    - array:包含该元素的数组对象
    - thisArg：可选参数，callbackfn 函数中的 this 关键字可引用的对象
  - 不会为数组中缺少的元素调用回调函数
  - map 将返回一个新数组
- filter() ：为数组的每个元素调用定义的回调函数，并返回毁掉函数为其返回 true 的值的数组
  - array.filter(callbackfn[,thisArg])
    - callbackfn：必须参数，最多可以接受 3 个参数的函数
    - value:数组元素的值
    - index 数组元素的数字索引
    - array:包含该元素的数组对象
    - thisArg：可选参数，callbackfn 函数中的 this 关键字可引用的对象
  - 不会为数组中缺少的元素调用回调函数

- 汇总：reduce() reduceRight()
  - array.**reduce**(callbackfn[,initialValue])
    - callbackfn：必须参数，最多可以接受 4 个参数的函数
    - previousValue:上一次调用回调函数的值
    - currentValue:当前数组元素的值
    - currentIndex:数组元素的数字索引
    - array:包含该元素的数组对象
    - initialValue：可选参数，如果有指定，则作为初始值来启动累积
  - 不会为数组中缺少的元素调用回调函数

函数

定义函数

使用函数

参 数

函数对象

This

闭包函数

案例分析

❄ 定义函数

♪ JavaScript 中定义函数的方法
- ✧ 使用 function 语句
  - ➢ function funName([args]){ statements }
    - ■ 声明语句是静态 function 结构
    - ■ var 和 function 都是变量声明语句!
    - ■ 声明的变量都在 JavaScript 预编译时被解析
    - ■ 解析器会把 function 语句定义为一个函数变量，函数体内所有参数、私有变量、嵌套函数作为属性注册到函数调用对象上
- ✧ 使用 Function()构造函数
  - ➢ var funName=new Function(p1,p2,…,pn,body);
    - ■ 参数类型都是字符串
    - ■ p1…pn，所创建函数的参数名称列表
    - ■ body，所创建函数的函数结构体语句
    - ■ Function 可以动态创建函数
    - ■ 它在执行期被编译，效率很低
- ✧ 定义函数直接量
  - ➢ function ([args]){ statements }
    - ■ 结构固定的函数体
    - ■ 也称 匿名函数

❄ 使用函数

♪ 函数两个实现与外界的交互的接口：
- ✧ 入口：参数
- ✧ 出口：返回值
  - ➢ 利用 return 关键之返回
  - ➢ return 可以返回一个函数

♪ 如果函数返回值为一个函数
- ✧ 则调用时可以使用多个小括号运算符反复调用
- ✧ EG:

```
function f(x, y){          // 定义函数
    return function(){     // 返回函数类型的数据
        return x * y;
    }
```

- }
- alert(f(7, 8)());　　// 返回值 56，反复调用函数

♪ 在嵌套函数中
- JavaScript 遵循从内到外的原则就近调用函数，但是不会从外到内调用函数。
- 这样就避免了嵌套函数中调用同名函数可能引发的冲突

❄ 参数

♪ 形参：
- 函数声明的参数变量

♪ 实参：
- 实际传递的参数值
  - arguments 对象：
    - 表示参数集合
    - 是一个伪类数组
    - 仅能够在函数体内使用
    - 可以通过修改其 *length* 值来增减参数
  - callee 回调函数：
    - callee 是 arguments 对象的一个属性
    - 其值是当前正在执行的 function 对象
    - 它的作用是使匿名 function 可以被递归调用
    - 如果不是匿名函数，则
      - arguments.callee 等价于函数名
  -

♪ JavaScript 的函数中，可以有若干个形参
- 函数定义时的形参可以通过 length 属性获取
- 如果函数实参数量少于形参数量
- 那么多出来的形参的值默认为 undefined
- 函数实参数量多于形参数量
- 如果那么多出来的实参就不能够通过形参标识符访问，函数会忽略掉多余的实参。
- 形参与函数体内使用 var 语句声明的变量
- 都属于局部变量，仅在函数体内可见
- 当私有变量与形参发生冲突时，则私有变量拥有较大的优先权

❄ 函数对象

♪ 获取实参个数：
- 使用 **arguments** 对象的 length 属性获取
- 仅能够在函数体内使用

- 获取形参个数：
  - 函数对象本身定义的 length 属性
  - 这个是一个只读属性
  - 与 arguments 对象的 length 属性不同，Function 对象的 length 属性可以在函数体**内外**都可以使用
- 使用 call()和 apply()
  - 两者是 Function 对象的原型方法
  - 它们能将特定函数当做一个方法绑定到指定对象上并进行调用
  - 语法格式：
    - function.call(thisobj,args…)
    - function.apply(thisobj,args)
  - ！
    - 使用call()和apply()方法可以把一个函数转换为指定对象的方法，并在这个对象上调用该方法。
    - 这种行为只是临时的，函数实际上并没有作为对象的方法而存在
    - 当函数被动态调用之后，这个对象的临时方法也会自动被注销
- bind()
  - 把函数绑定到指定对象上
  - function.bind(thisArg[,arg1[,arg2[,argN]]])
  - **this** 对象解析为传入的对象
  - 将返回与 **function** 函数相同的新函数

This

- 表示当前调用对象
- 函数调用 *VS* 函数引用
  - 引用：通常是指针指向
  - 调用：函数带()
  - 引用能够改变函数的执行作用域
  - 调用则不会改变函数的执行作用域
- call()**&**apply()
  - 直接改变函数的执行作用域
  - 使其作用域指向所传递的参数对象
- 原型继承
  - 两种情况
    - this 可能是指向子类的**实例对象**
    - this 也可能指向子类的**原型对象**
  - 函数闭包可能会对此造成影响
    - 使用方法的引用传递给父类的成员
    - 定义私有函数，在把它的引用传递给父类成员
      - 外界的其他对象都无法直接访问基类的私有函数
      - 对于相互依赖的方法，都可以定义为私有函数，以引用方法

- 异步调用
  - 定义：
    - 通过事件机制或者计时器来延迟函数的调用时间和时机
  - this：指向引发该事件的对象

- this 安全策略：
  - 避免把包含 this 的全局函数或方法动态用在局部作用域的对象中
  - 避免不同作用域的对象之间互相引用包含 this 的方法或属性
  - this 作为参数值调用函数
  - 设计 this 指针
    - 在构造函数中把 this 指针存储在私有变量中
    - 静态 this 指针

- 函数调用模式①②③
  - **方法调用模式**
    - 当一个函数被保存为对象的一个属性值时，称为方法
    - this 绑定到当前调用对象
  - **函数调用模式**
    - 当一个函数不是对象的属性时，被当做一个函数来调用——缺陷
    - this 绑定到全局对象
  - **构造器调用模式**
    - new 运算符调用，创建连接到 prototype 原型对象的新实例对象
    - this 绑定到新实例对象
  - **apply 调用模式**
    - 函数可以拥有方法，使用 apply 基本方法可以调用函数
    - this 绑定到 apply 传递的参数

❄ 闭包函数

- 什么是闭包函数
  - 定义：
    - 闭包函数就是嵌套结构的函数，在一个函数内定义的一个函数
    - 闭包函数的必要条件：
  - 内部定义函数
    - 内部函数应该访问外部函数中声明的私有变量、参数或者其他内部函数
    - 在外部函数**外**调用这个内部函数、
  - 如果没有闭包函数，则上例的数据结存和传递无法实施
    - 示例：
  - 闭包是函数运行期中的一个动态环境：
    - 闭包可以看成是函数的数据包，存储数据
    - 当函数调用返回之后，闭包保存着与函数关联变量的动态联系

- 使用闭包的情况
  - ①使用闭包结构能够跟踪动态环境中数据的实时变化，并即时存储
  - ②闭包不会因为外部函数环境的注销而消失，并始终存在

- ✧ ③利用闭包存储变量所有变化的值
- ✧ ④利用同一个闭包体声明多个闭包。

♪ 闭包的功能

- ✧ 为要执行的函数提供参数
- ✧ 创建额外作用域，由此设计动态数据管理器

♪ 特殊应用环境中使用闭包的优点

- ✧ 在事件处理中使用闭包：
  - ➢ 可以使用 JavaScript 函数来封装与特定 DOM 元素的交互 P194
    - ■ 不够优势：
    - ■ 如果闯将与不同 DOM 元素关联的任意数量的 JavaScript 对象，每个对象并不知道实例化它们的代码是如何操纵它们的
    - ■ 即：注册时间处理函数与定义相应的时间处理函数分离
- ✧ 使用闭包修正/提升之后：
  - ➢ 能够处理不同的 DOM 元素关联对象
  - ➢ 能够把时间函数与对象实例关联起来
  - ➢ 同时，传递参数！

❄ 案例

① 绑定函数

② 链式语法

③ 函数节流

④ 分支函数

⑤ 惰性载入函数

⑥ 惰性求值

⑦ 记忆

⑧ 构建模块

⑨ 柯里化

⑩ 高阶函数

⑪ 递归运算

⑫ 尾递归算法

对 象

创建对象

对 象

属 性

方 法

原 型

案例分析

❄ 1-创建对象

- 创建对象有 3 种方法：
  - 使用构造函数创建对象
  - 使用对象直接量创建对象
  - 使用 create()方法创建对象
- 使用**构造函数**创建对象
  - 语法格式：
    - `var objName = new funcName ( args ) ;`
  - 特点：
    - 使用 new 运算符进行调用
      - JavaScript 会自动创建一个空白的对象，然后把这个空对象传递给 this 关键字，作为它的引用值
    - 每个构造函数都定义了对象的一个类
    - 构造函数没有返回值——可以返回一个对象值
- 使用**对象直接量**创建对象
  - 语法格式：

```
var objName = {
    属性名：值，
    ……
};
```

  - 特点：
    - 最后一个属性末尾不需要逗号
    - 变量名是标识符，属性名是字符串标签
    - 对象的属性值可以是任意类型
- 使用 **create()方法**创建对象
  - 语法格式：
    - `Object.create(prototype,descriptors)`
  - 特点：
    - 这个是一个静态方法
    - 能够创建一个具有指定原型、且可选择性包含指定属性的对象
    - JavaScript 中，数据属性是可获取且可设置值的属性

❄ 2-对象

- 每个对象都拥有 3 个相关的对象特性：
  - 对象的原型 *prototype*
  - 对象的类 *class*
  - 对象的扩展标记 *extensible flag*
- **引用对象：**
  - 创建对象后，如果用赋值等式赋给变量
  - 则：
    - 是把对象的地址赋值给变量
    - 实现变量对对象的引用
- **复制对象：**
  - 利用 for/in 语句遍历对象成员，逐一复制给另一个对象
    - 可以利用对象直接量方法定义一个空对象
    - 对于复制，还可以通过原型对象扩展方法
  - Extend 方法不能够定义对象直接量，它只能够为构造函数结构复制对象
  - 复制操作要点：
    - 通过反射机制复制对象
    - 只能复制可枚举属性和方法——模拟继承
    - 效率很差
- **克隆对象：**
  - 方法：
    - 为 Function 对象扩展一个方法
  - 该方法能够把参数赋值给一个空构造函数的原型对象
  - 实例化构造函数
  - 返回实例对象——**该对象就拥有构造函数包含的所有成员**
  - 调用方法克隆对象
    - 优点：避免复制的低效率
- **销毁对象：**
  - 如果对象没有被任何变量引用时，JavaScript 会自动侦测，并运行垃圾回收程序把这些对象注销，释放内存。

❄ 3-属性

- 属性：名+值
- 属性特性：
  - 可写：
  - 可枚举：
  - 可配置：

- 属性操作：
  - **定义**
    - 语法格式：
    - 说明：
      - 可在结构体内定义属性
      - 可在结构体外定义属性：
        - 使用点运算符
      - 可以通过构造函数定义属性
    - 声明变量需要使用关键字 var，声明属性不可以使用 var
    - ECMAScript 5 增加了 2 个**静态函数**来指定对象定义属性：
      - Object.defineProperty
        - **Object.defineProperty(obj, propertyname, descriptor)**
      - Object.defineProperties
        - **Object.defineProperties(obj, descriptors)**
  - **访问**
    - 使用**点运算**访问
    - 使用**[“属性名”]**访问
    - 可以使用 for/in 遍历属性——可枚举自定义属性
      - 无固定的属性显示顺序
      - 无法枚举预定义属性
      - 读取不存在属性，返回 undefined
    - ECMAScript 5 新增 4 个函数访问对象属性：
      - Object.getPrototypeOf
      - Object.getOwnPropertyNames
      - Object.keys
      - Object.getOwnPropertyDescriptor
  - **赋值**
    - 可以通过赋值=操作更改对象属性值
    - 一旦为未命名的属性复制后，对象会自动创建该名称的属性
  - **删除**
    - 使用 **delete 运算符**删除对象属性
    - 当删除对象后：
      - 不是将该属性值设置为 undefined
      - 而是从对象中彻底清除属性——无法被枚举
  - **使用方法**
    - JavaScript 中，方法是对象属性的一种特殊形式
      - 值为函数的属性
    - 使用小括号运算符可以调用对象的方法
    - 在方法中，**this** 总是指向当前调用的对象
      - 对于构造对象：
        - 当对象实例化后，用户无法调用当前方法的实例对象名称
        - 使用 this 能确保在不同环境下都能找到调用当前方法的对象

♪ 配置属性

ECMAScript 5 增了 3 个函数来设置对象属性的特性:

✧ Object.seal：阻止修改现有属性的**特性**，阻止添加新属性

✧ Object.freeze：阻止修改现有属性的**特性**和**值**、或添加新属性

✧ Object.preventExtensions：阻止向对象添加新属性

♪ 检测特性

ECMAScript 5 增了 3 个函数来对对象属性特性进行检测：

✧ Object.isSealed：

✧ Object.isFrozen：

✧ Object.isExtensible:

❄ 4-方法

♪ 在 JavaScript 中，Object 对象默认定义了多个原型方法：

Object 基本方法	说　明
toString()	返回对象的字符串表示
toLocalString()	返回对象的本地字符串表示
valueOf()	返回对象的原始值
isPropotypeOf()	判断一个对象是否是另一个对象的原型
hasOwnProperty()	检查属性是否被继承
propertyIsEnumerable()	判断可否通过 for/in 循环遍历对象属性

♪ 1. toString()

✧ 可以对方法进行重写：

➢ 重写格式：

♪ Object.prototype.方法名=function(){……}

➢ 重写不会影响 JavaScript 内置对象的 toString () 返回值

✧ Object 对象定义的 toLocalString()方法默认返回值与 toString()方法完全相同

✧ 各不同对象的 toString()重写：

➢ Array：返回数组元素值的字符串组合

➢ Date：返回当前日期字符串表示

➢ Number：返回数字的字符串表示

♪ 2. valueOf()

✧ 默认情况下，Object 默认 valueOf()与 toString()返回值相同

- 部分类型对象重写了 valueOf()
  - String Number Boolean 返回其原始值
  - 自定义类型，最好重写 toString()和 valueOf()
- 某些环境下，valueOf()要比 toString()高级

♪ 3. hasOwnProperty()：检测私有属性

- 根据继承关系不同，对象属性分为：
  - 私有属性
  - 继承属性：构造函数的原型属性
- hasOwnProperty()
  - 只能判断指定对象中是否包含指定名称的属性
  - 无法检查对象原型链中是否包含某个属性

♪ 4.propertyIsEnumerable()：检测枚举属性

- 运算符 in 探测对象中是否存在某属性
  - in 会检测私有属性和原型属性
- 不是所有对象属性都可以枚举，只有用户自定义的本地属性和原型属性才允许枚举
- propertyIsEnumerable()：
  - 判断了**本地属性**是否允许枚举

♪ 5.isPropotypeOf()：检测原型对象

- JavaScript 中，Function 对象预定义了 propotype 属性
  - 指向一个原型对象
  - 当定义一个构造函数时，系统会自动创建一个对象，将其传递给 propotype
- 语法格式：
  - Obejct.propotype.isPropotypeOf(obj)

♪ 6.静态方法：

- JavaScript 核心对象中的 Math 和 Global 都是静态对象
- 类的静态成员分私有和公共两种类型
- 通过函数指针进行引用静态成员
- 在构造器内：关键字 var 声明私有成员和变量
- 在构造器内：关键字 this 声明公共成员和变量
- 在构造器内：不带 this 的都是静态方法
- 在构造器外：直接定义的都是**静态公共**属性和方法
  - 无法访问任何构造器中定义的**私有属性**

❄ 5-原型

♪ JavaScript 中，构造函数拥有原型

♪ 实例对象通过 prototype 可以访问原型

♪ 原型：P234

- 一个数据集合

- ✧ 即普通对象，继承自 Object 类
- ✧ 由 JavaScript 自动创建，并依附于每个构造函数

♪ 使用点语法，用户可以通过 function.prototype 方式定义原型，影响所有实例

♪ 原型属性 VS 本地属性：

- ✧ 如果本地属性与构造函数原型属性同名，则本地属性覆盖原型属性
- ✧ 本地属性可以在实例对象中被修改，且不同实例对象之间不会互相干扰
- ✧ 运行属性会影响所有实例对象

♪ 原型域与原型域链

- ✧ 在 JavaScript 中，实例对象读取属性时：
  - ➢ 由自己
  - ➢ →prototype 原型域
  - ➢ →由引用关系向外查找 prototype 原型域所指向对象的 prototype 原型域
  - ➢ →……
  - ➢ →找到它自己 或 循环 止
- ✧ 层层指向父原型的关系称为 **原型域链** Prototpye Chain

♪ 原型继承：

- ✧ JavaScript 中的继承：
  - ➢ 一切从对象的角度考虑
  - ➢ **直接定义对象，并被其他对象引用，就是一种继承关系**
  - ➢ 引用对象，称为**原型对象** Prototype Objcet
  - ➢ 可以通过原型域链来查找对象之间的继承关系

❄ 6-案例

♪ 工厂模式

♪ 类继承

第三章 JavaScript网络对象

❄ 计划课时：

18 学时

❄ 教学目的：

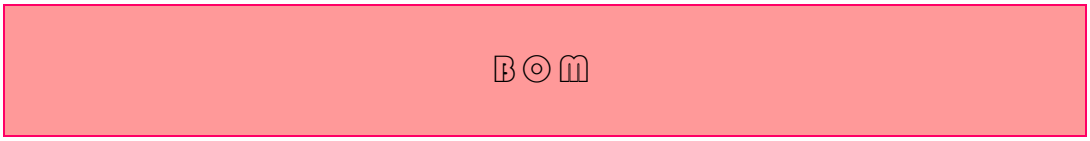
1. 认识 BOM
2. 掌握 DOM
3. 熟悉、熟练掌握事件响应模式

❄ 教学重点、难点：

1. 掌握：
   a) BOM 的各个元素控制、使用；
   b) DOM 结构、DOM 中主要方法的运用；
   c) 事件分类、事件模型、事件响应模式。
2. 熟悉：
   a) BOM 涉及的应用环境；
   b) DOM 的元素结构；
   c) 事件属性。
3. 了解：如何结合环境或要求，灵活整合所有 javascript 元素。

❄ 教学内容：

主要包含三大部分内容：BOM、DOM、事件响应



- BOM
  - Browser Object Model
  - 浏览器对象模型
  - 用于管理浏览器窗口

windows 对象

navigator 对象

location 对象

history 对象

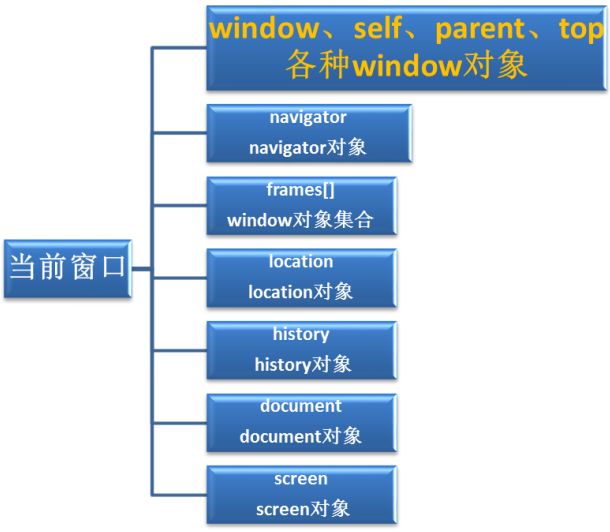
screen 对象

document 对象

案例分析

❄ windows 对象

- window 对象是 BOM 的核心
  - 代表浏览器窗口的一个实例
  - 它既是 JavaScript 访问浏览器窗口的接口，也是 JavaScript 的全局对象 Global
  - 在全局作用域中声明的所有变量和函数
    - →是 window 对象的属性和方法
- 访问浏览器窗口
  - 访问：
  - 通过 window 对象可以：
    - 访问浏览器窗口
    - 引用与浏览器相关的其他客户端对象——它们都是 window 的子对象
  - 客户端各个对象之间存在一种结构关系——即 浏览器对象模型：



  -
- 全局作用域
  - 客户端 JavaScript 代码都在全局上下文环境中运行
    - window 对象提供了 全局作用域
    - 所有全局变量都是为该对象的属性

- 使用 var 语句声明的全局变量：
  - window 会为其定义 configurable 特性
  - ——不能为 delete 运算符删除
- 未声明的全局变量：
  - JavaScript 会抛出异常

♪ 系统测试方法

- window 对象定义了 3 个人机交互的接口方法：
  - alert():提示对话框
  - confirm():“确定/取消”提示对话框
  - prompt():用户提示信息提示对话框
- 3 个方法仅接收纯文本信息，忽略 HTML 字符串
  - 用户只能使用空格 换行符 各种符号来格式化对话框中的显示文本
  - 不同浏览器显示效果可能不同
  - 可以重置方法
- 3 种方法打开对话框都是 同步 模态的
  - 在显示的时候，JavaScript 代码都会停止执行
  - 某些浏览器，如在 UNIX 下，alert 方法可能不产生暂停现象
- 3 种方法显示的外观由浏览器或操作系统决定，不由 CSS 决定

♪ 打开、关闭窗口

- 打开一个新窗口：
  - 新建的 window 对象拥有一个 opener 属性：
    - 保存着打开它的原始窗口对象
    - 在某些浏览器，当一个标签页打开另一个标签页时，如果两个 window 对象之间需要通信，则新的标签页就不能运行在独立的进程中
    - 如果将 opener 属性设置为 null，则新建的标签页就无法与打开它的标签页通信
    - 标签页之间的联系一旦切断，则无法恢复
- 关闭窗口：
  - 语法格式：
    - 关闭一个新建的 w 窗口：
      - w.close;
  - 在打开窗口内部关闭自身窗口：
    - window.close;
  - 使用 window.closed 属性可以检测当前窗口是否关闭：
    - true 关闭、 false 尚未

♪ 框架集

- 在 HTML 文档中，如果页面包含框架，则每个框架都拥有自己的 window 对象，并且，保存在 frames 集合中
- frames 集合：
  - 可以通过数字索引从左到右、从上到下访问每个 window 对象
  - 每个 window 对象都有一个 name 属性，包含框架名称
- 在每个框架中：

- window 对象:始终指向的都是那个框架实例，非最高层的框架
- top 对象:始终指向最高层的框架——浏览器窗口
- parent 对象：始终指向当前框架的上层框架

♪ 控制窗口位置

✧ window 对象的 screenLeft screenTop 属性可以读取和设置窗口相对于屏幕左边和上边的位置

- 用户无法跨浏览器获取窗口属性值

✧ window 对象的 moveTo() moveBy()可以将窗口精确地移动到下一个新位置

♪ 控制窗口大小

✧ window 对象的 4 个属性可以确定窗口大小：

- innerWidth/innerHeight：
- outerWidth/outerHeight：

✧ 在 IE、Chrome 中，有 2 属性保存页面视图信息：

- doucument.documentElement.clientWidth
- doucument.documentElement.clientHeight

♪ 定时器

✧ window 对象包含 4 个定时器专用方法:

- setTimeOut()
  - 在指定时间段后执行特定代码
  - 语法格式：
    - var id = setTimeOut (code, delay) ;
    - clearTimeOut(id);
  - 特点：
    - 只执行一次
- clearTimeOut()
- setInterval()
  - 周期性执行指定代码
  - 语法格式：
    - var id = setInterval (code, delay) ;
    - clearInterval(id);
  - 特点：
    - 还可指定函数参数运行
- clearInterval()

❄ *navigator 对象*

♪ 包含了浏览器的基本信息；通过 *window.navigator* 可以引用该对象

♪ 浏览器检测方法：

✧ 特征检测法：

- 根据浏览器是否支持特定功能来决定操作的方式
- 当使用一个对象、方法、属性时，先判断它是否存在——即浏览器是否支持它

- 字符串检测法：
  - 客户端浏览器每次发送 HTTP 请求时，都会附带一个 user-agent 字符串，通过它，web 开发人员可以识别客户使用的浏览器类型

- 检测浏览器类型和版本号：
  - 只需要根据不同浏览器类型匹配特殊信息即可
  - 通过解析 navigator 的 userAgent 属性，可以获得浏览器的完整版本号

location 对象

- location 对象存储当前页面与位置 URL 相关的信息
  - 表示当前显示文档的 web 地址
  - 通过 *window.location* 属性可以访问

- location 定义了 8 个属性：
  - Href Protocol Host Hostname Port Pathname Search Hash
  - location 对象的属性都是可读可写的
    - 只要修改 href，则会加载入新的页面
  - location 对象还定义了两个方法：
    - reload() 和 replace()

history 对象

- history 对象存储浏览器窗口的浏览历史
  - 通过 window.history 属性可以访问该对象
  - history 对象进制 JavaScript 脚本直接操作它的访问信息
  - 方法：
    - back()
    - forward()
    - go()
  - 每个窗口都有独立的历史记录，并通过独立的 history 属性引用

- screen 对象

document 对象

- 访问文档对象：
  - 浏览器加载文档时，会自动构建文档对象模型
    - 把文档中同类元素对象映射到一个集合中
      - 这些集合都是 HTMLCollection 对象
    - 然后，以 document 对象属性的形式允许用户访问

- 动态生成文档内容：
  - 使用 document 对象的 write()和 writeln()方法可以动态生成文档内容

案例分析

DOM

DOM 基础

节 点

文档节点

元素节点

文本节点

文档片段节点

属性节点

范围

案例分析

❄ DOM 的发展

♪ W3C 对 DOM 进行了标准化
- ✧ DOM 1 级：主要包含 2 个子规范：
  - ➢ DOM Core
  - ➢ DOM HTML
- ✧ DOM2 级：主要包含 6 个子规范：
  - ➢ DOM2 Core
  - ➢ DOM2 HTML
  - ➢ DOM2 Event
  - ➢ DOM2 Style
  - ➢ DOM2 Travelsal & DOM2 Range
  - ➢ DOM2 Views
- ✧ DOM 3 级：主要包含 3 个子规范：
  - ➢ DOM3 Core
  - ➢ DOM3 Load and Save
  - ➢ DOM2 Validation

❄ 节 点

♪ DOM 1 级定义了 Node 接口

♪ JavaScript 实现了这个接口
- ✧ 定义：

- 所有节点类型必须继承 Node 类型
- 所有 Node 的子类或孙类，都拥有 Node 的基本属性和方法

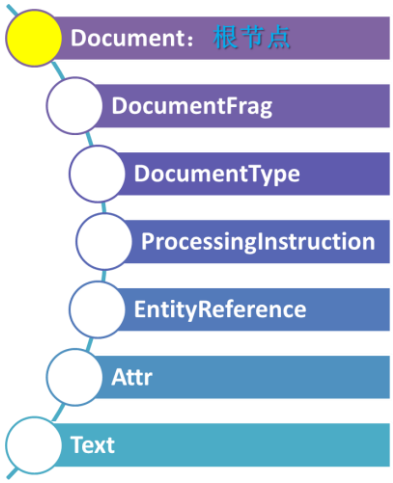
♪ 节点类型：

DOM 规定：

- ✧ 整个文档就是一个文档节点
- ✧ 每个标签是一个元素节点
- ✧ 元素节点包含的文本是文本节点
- ✧ 元素的属性是一个属性节点
- ✧ 注释属于注释节点
- ✧ ……

每个节点都有一个 nodeType 属性：

- ✧ 表明节点的类型



✧

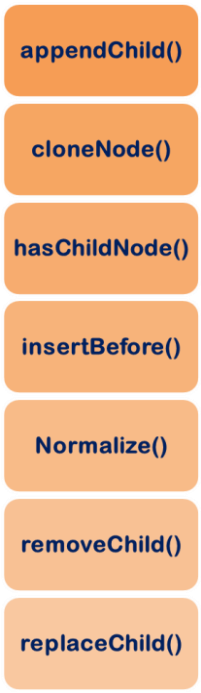
♪ 节点名称 nodeName 和值 nodeValue

♪ 节点关系：

- ✧ DOM 把文档视为一种树结构——称为 节点树

♪ 访问节点：

- ✧ DOM 为 Node 类型定义如下属性：
  - ownerDocument:
  - parentNode:返回元素类型节点
  - childNodes:保存一个 nodeList 对象
  - firstNode:
  - lastNode:
  - nextSibling:
  - previousSibling:



- 操作节点：
  - appendChild insertChild removeChild replaceChild
    - 并不是所有的类型的节点都有子节点
    - 不支持子节点的节点不可以调用上述 4 个方法
  - cloneNode
    - 复制后返回的节点副本属于文档所有
    - 但并未为其指明父节点
    - 不会复制添加到 DOM 节点中的 JavaScript 属性
    - 这个方法值复制 HTML 特性或子节点

❄ 文档节点

- 在 DOM 中：
  - Document 类型：表示文档节点
  - HTMLDocument：是 Document 的子类
  - document 对象：HTMLDocument 的实例
    - 表示 HTML 文档
    - 也是 window 对象的属性，可在全局作用域中直接访问

- Document 节点特征：
  - nodeType 值：9
  - nodeName 值：#document
  - nodeValue 值：null
  - parentNode 值：null
  - ownerDocument 值：null
  - 可能的子节点：
    - DocumentType

- Element
- ProcessingInstruction
- Comment

♪ 访问文档子节点：
- ✧ 方法：
  - 使用 documentElement 属性
  - ——始终指向 HTML 页面中的 html 元素
  - 使用 childNodes 列表
- ✧ 所有浏览器都支持 document.documentElement 和 document.body 用法
- ✧ <!DOCTYPE>标签：
  - 是一个与文档主体不同的实体
  - 可以通过 document.doctype 属性访问它

♪ 访问文档信息：
- ✧ HTMLDocument 的实例对象 document 包含很多属性来访问文档信息：
  - title：
  - lastModified：
  - URL：
  - domain：
  - referrer：

♪ 访问文档元素：
- ✧ document 对象有许多访问文档内元素的方法：
  - getElementById()：
  - getElementsByTagName()：
    - ■ 返回一个 HTMLCollection 对象
      - ◆ 它含有一个 namedItem()方法：通过元素的 name 特性取得集合中的项目
  - getElementsByName()：

♪ HTML5 Document
- ✧ readyState：
- ✧ compatMode：
- ✧ head：
- ✧ charset：
- ✧ defaultCharset：

❄ 元素节点

♪ 节点 Element 类型特征：
- ✧ nodeType 值：1
- ✧ nodeName 值：元素的标签名
- ✧ nodeValue 值：null
- ✧ parentNode 值：Document 或 Element 类型节点
- ✧ 可能的子节点：

- Element
- Text
- Comment

- ProcessingInstruction
- CDATASection
- EntityReference

- 操作
  1. 访问元素
  2. 遍历元素
  3. 创建元素
  4. 复制节点
  5. 插入节点
  6. 删除节点
  7. 替换节点
  8. 获取焦点元素
  9. 检测包含节点

- 访问元素：
  - getElementById()：
    - 获取文档中指定元素
    - 该方法只适用于 document 对象
  - getElementByTagName()：
    - 获取指定标签名称的所有元素
    - 返回值：一个节点集合

- 创建元素：
  - 语法格式：
    - **var e=document.createElement(“tagName”);**
    - 根据参数指定的标签名称创建一个新的元素
  - 该方法创建的新元素不会被自动添加到文档里：
    - 因为新的元素的 nodeParent 属性没有指定
    - 需要使用 appendChild、insertBefore 或 replaceChild 方法来添加

- 复制节点：
  - cloneNode（）方法
    - 复制节点会包含原节点的所有特性
    - 特别需要注意 id 属性的重复情况

- 插入节点：
  - appendChild()
    - 在当前节点的子节点列表的末尾添加新的子节点
  - insertBefore()
    - 在已有的子节点钱插入一个新的子节点

- 删除节点：
  - removeChild()

- 删除的节点，其包含的所有子节点将同时被删除

♪ 获取焦点元素：
- ✧ HTML5xinzeng DOM 焦点管理功能
  - ➢ 使用 document.activeElement 属性：
  - ➢ 可以引用 DOM 中当前获得了焦点的元素
- ✧ 元素获取焦点的方式：
  - ➢ 页面加载
  - ➢ 用户输入
  - ➢ 在脚本中调用 focus()方法

♪ 检测包含节点：
- ✧ contains()
- ✧ *IE*的私有方法
- ✧ 加测某个节点是不是另一个节点的后代

❄ 文本节点

♪ 文本节点：
- ✧ 由 Text 类型表示
  - ➢ 包含纯文本内容，或转义后的 HTML 字符，但不能包含 HTML 代码
- ✧ 特征：
  - ➢ nodeType 值：3
  - ➢ nodeName 值：#text
  - ➢ nodeValue 值：节点所包含的文本
  - ➢ parentNode 值：是一个 Element 类型节点
  - ➢ 可能的子节点：不包含

♪ 访问文本节点：
- ✧ 访问 Text 节点中包含的文本方法：
  - ➢ 使用 nodeValue 属性
  - ➢ 使用 data 属性
- ✧ 文本节点包含 length 属性

♪ 创建文本节点：
- ✧ 语法格式：
  - ➢ Document.createTextNode(data)

♪ 操作 文本节点
- ✧ appendData(string)
- ✧ deleteData(start,length)
- ✧ insertData(start,string)
- ✧ repalceData(start,length,string)
- ✧ splitText(offset)
- ✧ substringData(start,length)
- ✧ 默认情况下，

- 每个可以包含内容的元素最多只能有一个文本，且须确保有内容存在

♪ 读取 HTML 字符串
- ✧ 元素的 innerHTML 属性可以返回
  - ➢ 调用元素
    - ■ 包含的所有子节点
      - ◆ 对应的 HTML 标记字符串

♪ 插入 HTML 字符串
- ✧ innerHTML 属性
  - ➢ 它可以根据还如的 HTML 字符，创建新的 DOM 片段
  - ➢ 然后，用 DOM 片段完全替换调用元素原有的所有子节点
- ✧ insertAjacentHTML()
  - ➢ 插入 HTML 标记的另一种新增方式
  - ➢ 被 HTML5 规范化了
  - ➢ 其参数都是小写形式

♪ 替换 HTML 字符串
- ✧ 在读模式下：
  - ➢ outerHTML 返回调用它的元素及所有子节点的 HTML 标签
- ✧ 在写模式下：
  - ➢ outerHTML 根据指定的 HTML 字符串创建新的 DOM 子树
  - ➢ 然后用这个 DOM 子树完全替换调用元素

♪ 插入文本
- ✧ innerText
  - ➢ 在指定元素中插入文本内容
- ✧ outerText
  - ➢ 能覆盖原有的元素

❄ 文档片段节点

♪ DocumentFragment
- ✧ 在文档树中没有对应的标记
- ✧ 特征：
  - ➢ nodeType 值：11
  - ➢ nodeName 值：#document-fragment
  - ➢ nodeValue 值：null
  - ➢ parentNode 值：null
- ✧ 可能的子节点：
  - ➢ Element，ProcessingInstruction，Comment，Text，CDATASection，EntityReference

♪ DocumentFragment 作用：
- ✧ 将文档片段作为节点“仓库”来使用，保存将来可能会添加到文档中的节点
- ✧ 语法格式：

- var fragment=document.createDocumentFragment();
- 如果将文档树中的节点添加到文档片段，则文档树中会移除该节点
- 添加到文档片段的节点同样也不属于文档树
- 使用 appendChild（）或 insertBefore（）方法可以将文档片段添加到文档树

✧ 文档片段本身永远不会成为文档树的一部分

❄ 属性节点

♪ 属性节点：

✧ Attr 类型表示
- 在文档树中，称为元素的特性或标签的属性

✧ 特征：
- nodeType 值：11
- nodeName 值：特性的名称
- nodeValue 值：特性的值
- parentNode 值：null

✧ 可能的子节点：
- HTML 中不包含子节点
- XML 中子节点可以是 Text，EntityReference

♪ 访问属性节点：

✧ Attr 没有父节点，是 Element 的属性，继承了 Node 的属性和方法

✧ Attr 对象 3 个专用属性：
- name：属性名称
- value：属性的值
- specified：

✧ 创建属性节点的语法格式：
- **document.createAttribute(name)**

✧ 将新创建的属性添加到元素中，语法格式：
- **元素对象.setAttributeNode(属性节点)**

♪ 读取属性值：

✧ 使用元素的 getAttribute()方法
- 可快速读取指定元素的属性值

✧ 使用元素的点语法也可以快捷读取属性值

♪ 设置属性值：

✧ 元素设置属性语法格式：
- **setAttribute(name,value)**

✧ DOM 支持使用 getAttribute（）和 setAttribute（）方法读写自定义属性

♪ 删除属性：

✧ 元素删除属性语法格式：
- **removeAttribute(name)**

♪ 使用类选择器：

- HTML5 为 document 对象和 HTML 元素新增了一方法
  - getElementByClassName（）
    - 可以选择指定类名的原色
    - 返回带有指定类的所有元素的 NodeList

♪ 自定义属性：
- HTML5 允许用户为元素自定义属性
  - 需要添加前缀：“data-”
- 目的：
  - 为元素提供与渲染无关的附加信息
  - 或 提供语义信息
- 访问：
  - 通过元素的 dataset 属性可以访问
  - dataset 属性的值是一个 DOMStringMap 实例
    - 即 名值对 的映射
  - 每个 data-name 形式的属性都会有一个对应的属性
    - 属性名前，没有前缀 data-

❄ 范围

♪ 创建范围：
- 定义语法：
  - var range = document. createRange ()
- 特点：
  - 每个范围：都有 2 个边界点（一个**开始点**和一个**结束点**）
  - 每个边界点：由一个**节点**和该节点的**偏移量**指定
    - 该节点通常是 Element 节点、Document 节点 或 Text 节点
      - **Element 节点、Document 节点**：偏移量是该节点的子节点
      - **Text 节点**：偏移量是文本中两个字符之间的位置
- 范围：
  - 实际上就是 **Range 类型**的一个实例对象

♪ 选择范围：
- 创建范围之后，可以选择文档中的某一部分：
- **selectNode()和 selectNodeContents()方法**
- **range.selectNode(node)**
  - 参数为 node 节点，把整个 node 节点的信息包括子节点中的内容填充到范围 range 内。
- **range.selectNodeContents(node)**
  - 参数为 node 节点，把 node 节点的**子**节点信息填充到范围 range 内。

♪ 设置范围：
- 方法：
- setStart()方法 和 setEnd()方法

♪ 操作范围内容：

- 范围实际上是一个文档片段
  - 创建范围之后，其内全部节点都被添加到这个文档片段中
  - 文档片段会自动补全开始标签和结束标签，重构有效的 DOM 结构
- 使用范围的 deleteContents()方法能够从文档中删除范围所包含的内容
- 使用范围的 extractContents()方法能够把范围一致到文档的其他位置
- 使用范围的 cloneContents()方法能够把复制范围内容

♪ 插入范围内容：
- 使用范围的 insertNode()方法能够向范围开始位置插入一个节点
- 也可以使用范围的 surroundContents()方法给范围包裹一个节点

♪ 复制和清除范围：
- 使用范围的 cloneContents()方法能够把复制范围
- 用完范围之后，最好使用范围的 detach()方法把范围从文档中分离出来
  - 这样可以便于解除引用
  - 垃圾回收内存

❄ 案例分析

事件处理

事件基础

鼠标事件

键盘事件

页面事件

UI 事件

案例分析

❄ 事件基础

♪ 事件模型：
- 发展历史：
  - 基础事件模型：DOM0
  - DOM 事件模型：
  - IE 事件模型：
  - Netscape 事件模型：

♪ 事件流：
- 概念：就是多个节点对象对同一种事件进行相应的先后顺序
- 事件流类型：

- 冒泡型：
  - 从最特定的目标向最不特定的目标 document 对象依次触发事件，从下向上响应
- 捕获型：
  - 从最不特定的目标开始被触发，最后到最特定的目标，从上向下响应
- 混合型：
  - 支持两种事件流，但捕获型事件流先发生，后发生冒泡型事件流
  - 两种事件流会触及 DOM 中的所有层级对象:从 document 对象开始，最后返回 document 对象
  - 三个阶段：
    - **捕获阶段→目标阶段→冒泡阶段**

♪ 事件类型:

- ✧ DOM0 将浏览器中发生事件类型分成依据触发对象不同：
  - 鼠标事件：跟踪鼠标的当前定位事件、跟踪鼠标的点击事件
  - 键盘事件：keyup、keydown、keypress
  - 页面时间：页面本身的行为
  - UI 事件：用户在页面中的行为
- ✧ DOM2 事件模块包含 4 个子模块每个子模块提供对某类事件的支持：
  - HTMLEvents:
  - MouseEvents:
  - UIEvents:
  - MutationEvents:其定义的事件在文档改变时生成的，不常用

♪ 绑定事件：

- ✧ 在基本事件模型中，JavaScript 支持两种绑定方式：
- ✧ 静态绑定：
  - 把 JavaScript 脚本作为属性值，直接赋予事件属性
- ✧ 动态绑定：
  - 使用 DOM 对象的事件属性进行赋值

♪ 事件处理函数：

- ✧ 定义：
  - 是一类特殊的函数，用于实现事件处理；一般没有明确的返回值
  - 是异步调用，由事件触发进行响应
- ✧ 特点：
  - 事件处理函数不需要参数——默认包含 event 参数对象
  - IE 事件模型与 DOM 事件模型对于 event 对象的处理方式不同：
    - IE 把 event 对象定义为 window 对象的一个属性 VS
    - DOM 把 event 定义为事件处理函数的默认参数
  - 事件处理函数中的 this 表示当前事件对象

♪ 注册事件：

- ✧ DOM 事件模式中：
  - 调用 addEventListener()方法注册事件

- 语法格式：
- e.addEventListener(String t, Function l, boolean useCapture)
- t：注册事件的类型名，事件类型名前没有 on 前缀
- l：处理函数
- useCapture：选择在捕获阶段或冒泡阶段触发
- 该方法能够为多个对象注册相同的事件处理函数，也可以为同一个对象注册多个事件处理函数

✧ IE 事件模式中：

- 使用 e.attachEvent(etype,eventName)注册事件

♪ 销毁事件：

✧ DOM 事件模式中：

- 调用 removeEventListener()方法删除已经注册的事件处理函数
- 语法格式：
- e.removeEventListener(String t, Function l, boolean useCapture)

✧ IE 事件模式中：

- 使用 e.detachEvent(etype,eventName)注册事件

♪ 使用 event 对象：

✧ event 对象由事件自动创建，代表事件的状态：

- 它的属性提供了有关事件的细节
- 它的方法可以控制事件的传播
- 图示对比➔

✧ P374

- 以事件驱动为核心的设计中，一次只能处理一个事件
  - 建议：使用全局变量来存储事件信息——比较安全

✧

♪ 事件委托：

✧ delegate 也称：事件托管 或 事件代理

✧ 优点：

- 防止在动态添加或删除节点的过程中注册的事件丢失

❄ 鼠标事件

♪ 鼠标事件概述：

✧ click

✧ dbclick

✧ mousedown

- mouseover
- mouseup
- mousemove

♪ 鼠标点击：
- 包括 4 个事件：
  - click
  - dbclick
  - mousedown
  - mouseup
- 当这些事件处理函数的返回值为 false 时，会禁止捆绑对象面对默认行为

♪ 鼠标移动：
- mouseover 事件类型是一个实时响应的事件
  - 事件响应的灵敏度主要参考鼠标指针移动速度的快慢 以及 浏览器跟踪跟新的速度

♪ 鼠标经过：
- 包括 移过 和 移出 两种事件类型
- 如果从父元素中移动子元素中，也会触发父元素的 mouseover 事件类型

♪ 鼠标来源：
- 一个事件发生后，使用事件对象的 target 属性，获取发生事件的节点元素
  - 如果是 IE 事件模型中实现相同的目标，可以使用 srcElement 属性
- DOM 模型中，还定义了 currentTarget 属性
  - 故此，一般事件处理函数中，应该使用该属性而不是 this 关键字
  - 否则，可能执行设计者意外的对象

♪ 鼠标按键：
- 通过实践对象的 button 属性可以获取当前鼠标按下的键
  - 该属性可以用于 click mousedown mouseup 事件类型
- IE 模式支持 位掩码 技术，能够侦测到同时按下的多个键
- VS
- DOM 模式不支持掩码技术

❄ 键盘事件

♪ 概述：
- 键盘事件主要包括 3 种类型：
  - keydown：该事件处理函数返回 false 时，会取消默认的动作
  - keypress：该事件处理函数返回 false 时，会取消默认的动作
  - keyup：该事件不支持一个持续的响应状态
- 几乎所有的元素都支持键盘事件，但，键盘事件多被应用在表单输入中。

♪ 键盘事件属性：
- 键盘事件属性一般只在键盘相关事件发生时才会存在于事件对象中
  - ctrlKey、shiftKey 属性除外——它们也存在在鼠标事件中

- 属性：
  - keyCode charCode target srcElement shiftKey ctrlKey altKey metaKey
- 需要注意：
  - keyCode 属性 VS charCode 属性
  - 在不同事件类型和不同浏览器中的表现有很多区别
- P386

♪ 键盘响应顺序：

- 当按下键盘键时，会 **连续**触发 **多** 个事件，它们将按顺序发生。
  - 字符键事件响应顺序 ：keydown → keypress → keyup
  - **非**字符键事件响应顺序：keydown → keyup

❄ 页 面 事 件

♪ 概述：

- 所有的页面事件都明确地处理整个页面的函数和状态
- 主要包括：页面的加载和卸载
  - 即：用户访问页面和离开关闭页面的事件类型

♪ 页面初始化：

- load 事件类型：在页面完全加载完毕的时候触发
- 为 window 对象绑定 load 事件类型的方法：
  - 直接为 window 对象注册页面初始化事件处理函数
  - 在页面<body>标签中定义 onload 事件处理属性
- load 事件类型经常需要调用附带参数的函数：
  - 在 body 元素中，通过事件属性的形式调用函数
  - 通过函数嵌套或闭包函数实现

♪ 结构初始化：

- DOMContentLoaded 事件类型：
  - DOM 标准事件
  - 在 DOM 文档结构加载完毕时触发，比 load 事件类型先被触发

♪ 页面卸载：

- unload：
  - 事件在从当前浏览器窗口内移动文档的位置时触发
- 在 unload 事件中，无法有效阻止默认行为
- beforeunload 事件类型：
  - 可以返回任意类型值

♪ 窗口重置：

- resize 事件类型：
- 在浏览器窗口被重置时触发

♪ 页面滚动：

- scroll 事件类型：

- 在浏览器窗口内移动文档的位置时触发

♪ 错误处理：
- ✧ error 事件类型：
  - ➢ JavaScript 代码发生错误时触发
  - ➢ 无须传递事件对象，且包含已经发生错误的解释信息

❄ 2.7 事件

♪ 焦点处理：
- ✧ 焦点：激活表单字段，使其可以相应键盘事件
- ✧ 焦点处理：
  - ➢ focus 事件类型：
  - ➢ blur 事件类型：
- ✧ 注意：
  - ➢ 如果隐藏字段,或使用 CSS 的 display 和 visibility 隐藏字段显示，则获取焦点会引发异常

♪ 选择文本：
- ✧ select 事件：
  - ➢ 用户选择了文本，且要释放鼠标，才会触发

♪ 字段值变化监测：
- ✧ change 事件：
  - ➢ 在表单元素的值发生变化时触发
  - ➢ 主要用于 input、select、textarea 元素

♪ 提交表单：
- ✧ submit 事件类型：
  - ➢ 仅在表单内单击提交按钮、或在文本框中输入文本时按回车键 触发
- ✧ 注意：
  - ➢ 在<textaere>文本区中的回车键不会触发事件，只是换行
  - ➢ 阻止事件的默认行为可以取消表单提交

♪ 重置表单：
- ✧ reset 事件：
  - ➢ <input>、<button>标签的属性设置“type=“reset””重置触发

♪ 剪贴板数据：
- ✧ 6 个主要剪贴板事件：
  1. beforecopy
  2. copy
  3. beforecut
  4. cut
  5. beforepaste
  6. paste

❄ 案例分析

