第五周

Day20笔记

|  |
| --- |
| **1. 操作符**  1.1 算术运算符   * **加法 (+)**: 数字相加，字符串拼接。   var num = 1 + 2; // 3  var str = "Hello" + " " + "World"; // "Hello World"   * **减法 (-)**: 数字相减，可以将字符串转换成数字后相减。   var num = 4 - 3; // 1  var num = "8" - "4"; // 4   * **乘法 (\*)**: 数字相乘，可以将字符串转换成数字后相乘。   var num = 4 \* 3; // 12   * **除法 (/)**: 数字相除，可以将字符串转换成数字后相除。   var num = 6 / 3; // 2   * **取余 (%)**: 取余数操作。   var num = 7 % 3; // 1  1.2 一元运算符   * **正号 (+)**: 将操作数转换成数字。   var num = +"123"; // 123   * **负号 (-)**: 将操作数转换成负数。   var num = -"123"; // -123   * **逻辑取反 (!)**: 将布尔值取反。   var bool = !true; // false   * **递增 (++)**: 变量自增1。   var num = 1;  num++; // 2   * **递减 (--)**: 变量自减1。   var num = 1;  num--; // 0   * **删除 (delete)**: 删除对象或数组的属性。   var obj = { name: 'John', age: 30 };  delete obj.age; // { name: 'John' }   * **类型判断 (typeof)**: 返回操作数的类型。   var type = typeof 123; // "number"  **2. 赋值运算符**   * **赋值 (=)**: 赋值操作。   var x = 5;   * **加等 (+=)**: 相加后赋值。   var x = 5;  x += 2; // 7   * **减等 (-=)**: 相减后赋值。   var x = 5;  x -= 2; // 3   * **乘等 (\*=)**: 相乘后赋值。   var x = 5;  x \*= 2; // 10   * **除等 (/=)**: 相除后赋值。   var x = 5;  x /= 2; // 2.5   * **取余等 (%=)**: 取余后赋值。   var x = 5;  x %= 2; // 1  **3. 比较运算符**   * **等于 (==)**: 仅值相等即可。   var result = (5 == "5"); // true   * **全等 (===)**: 值和类型都要相等。   var result = (5 === "5"); // false   * **不等于 (!=)**: 值不等即可。   var result = (5 != "5"); // false   * **不全等于 (!==)**: 值或类型不等。   var result = (5 !== "5"); // true   * **大于 (>)**: 大于。   var result = (5 > 3); // true   * **小于 (<)**: 小于。   var result = (5 < 3); // false   * **大于等于 (>=)**: 大于或等于。   var result = (5 >= 3); // true   * **小于等于 (<=)**: 小于或等于。   var result = (5 <= 3); // false  **4. 隐式转换**   * **ToPrimitive**: 将值转换为原始值。 * **ToNumber**: 将值转换为数字。 * **ToString**: 将值转换为字符串。   **隐式转换规则**   1. 字符串与数字相加时，数字会被转换为字符串。   var result = 1 + "2"; // "12"   1. 字符串与数字相减时，字符串会被转换为数字。   var result = "5" - 3; // 2  **5. 流程控制语句**   1. **if 语句**: 条件判断。   if (condition) {  // TODO  }   1. **switch 语句**: 多条件判断。   switch (expression) {  case value1:  // TODO  break;  case value2:  // TODO  break;  default:  // TODO  }   1. **for 循环**: 计数器控制的循环。   for (var i = 0; i < 5; i++) {  // TODO  }   1. **while 循环**: 条件控制的循环。   j  while (condition) {  // TODO  }   1. **do...while 循环**: 至少执行一次的循环。   do {  // TODO  } while (condition);  **使用注意**   * 尽量使用严格比较运算符（===），避免隐式转换带来的困惑。 * 注意类型转换的规则和顺序，避免意外的结果。   日期2024.7.22 |

Day21笔记

|  |
| --- |
| **1. JavaScript 对象基础**   * **对象定义**: JavaScript中的对象是无序属性的集合，其属性可以包含基本值、对象或函数。对象是由键值对（key   ）组成，值可以是数据或函数。   * **现实中的对象类比**: 每个人都是一个对象，具有属性（如身高和体重）和方法（如走路和跑步）。每个人的属性和方法虽相似，但具体值和执行时间各不相同。 * **对象创建**:   + **字面量模式**: 使用 {} 作为对象的边界，属性之间用 , 隔开，属性名和属性值用 : 隔开。   + **构造函数模式**: 使用 Object 或自定义构造函数来初始化对象。   var person = {name:"张三", height:1.71, gender: 'male'};  var obj = new Object();  obj.name = "terry";  obj.age = 12;  obj.sayName = function() {  console.log("my name is", this.name);  };  **2. 对象的访问**   * **属性访问**: 有点访问（.）和中括号访问（[]）两种方式。   console.log(obj.name);  console.log(obj['name']);   * **方法的访问**: 需要按照函数调用的方式使用对象的方法。   obj.sayName(); // 调用对象的方法  **3. 遍历对象的属性**   * 使用 for..in 循环遍历对象的属性。   for (var key in obj) {  console.log(key + "--" + obj[key]);  }  **4. 新增和删除对象中的属性**   * 新增属性可以直接赋值。   obj.newProperty = "newValue";   * 删除属性使用 delete 关键字。   delete obj.propertyName;  **5. 对象的强制类型转换**   * **转换为 Boolean**:   var obj = { name: "briup", age: 12 };  console.log(Boolean(obj)); // true   * **转换为 String**: 调用 toString 方法。   console.log(obj.toString());   * **转换为 Number**: 调用 valueOf 方法。   var obj = {  valueOf: function() { return 10; }  };  console.log(Number(obj)); // 10  日期2024.7.23 |

Day22笔记

|  |
| --- |
| **开题报告、开题PPT，任务书，选题审批表**  **1. 检测属性**   * **in 运算符**: 检测属性是否存在于对象中，无论是自有属性还是继承属性。   console.log('name' in obj); // true   * **hasOwnProperty**: 检测属性是否为对象的自有属性。   console.log(obj.hasOwnProperty('name')); // true   * **propertyIsEnumerable**: 检测属性是否为对象的可枚举自有属性。   console.log(obj.propertyIsEnumerable('name')); // true  **2. 对象原型属性及方法**   * **Object 构造函数的原型属性和方法**: 包括 constructor、hasOwnProperty、propertyIsEnumerable、valueOf、toLocaleString、toString 和 isPrototypeOf。   var obj = new Object();  console.log(obj.toString());  console.log(Object.prototype);  **3. 深入理解对象-定义属性**   * **数据属性**: 包含 [[Configurable]]、[[Enumerable]]、[[Writable]] 和 [[Value]] 特性。   Object.defineProperty(obj, 'name', {  configurable: true,  enumerable: true,  writable: true,  value: 'terry'  });   * **访问器属性**: 包含 [[Configurable]]、[[Enumerable]]、[[Get]] 和 [[Set]] 特性。   Object.defineProperty(obj, 'fullName', {  get: function() { return this.firstName + ' ' + this.lastName; },  set: function(value) {  var parts = value.split(' ');  this.firstName = parts[0];  this.lastName = parts[1];  }  });  日期2024.7.24 |

Day23笔记

|  |
| --- |
| **函数概述与声明**  **1. 函数介绍**   * **定义**：函数是封装一系列代码以完成特定任务的结构。 * **用途**：   + 功能封装，提高代码复用率   + 构建对象的模板（构造函数） * **函数类型**：每个函数都是 Function 类型的实例，具有属性和方法。   **2. 函数声明与表达式**   * **函数声明**：   function 函数名(形参列表) {  // 函数体  }  示例：  function add(a, b) {  var result = a + b;  return result;  }  var total = add(1, 2); // total = 3   * **函数表达式**：   var 函数名 = function(形参列表) {  // 函数体  }  示例：  var add = function(a, b) {  var result = a + b;  return result;  }  **3. 函数提升**   * 函数声明会被提升到代码的最前面，可以在声明前调用。   foo(); // 可以调用  function foo() {  console.log("Hello World");  }   * 函数表达式不会提升，必须在赋值后调用。   **4. 函数内部属性**   * **arguments**：类数组对象，包含传入函数的所有参数。   + arguments.callee：指向拥有该 arguments 对象的函数。   function add(a, b) {  console.log(arguments[0], arguments[1]);  return a + b;  }  add(10, 20); // 输出：10 20  **5. this 关键字**   * **用法**：   + 方法中：this 指向方法所属的对象。   + 函数中：默认指向全局对象（浏览器中为 window，Node.js 中为 global）。   + 事件中：指向接收事件的元素。   + 显式绑定：使用 call 和 apply 切换 this 的上下文。   var person = {  firstName: "张",  lastName: "建安",  fullName: function() {  return this.firstName + " " + this.lastName;  }  };  console.log(person.fullName()); // 输出  **1. IIFE（立即调用函数表达式）**   * **定义**：声明函数的同时立即调用该函数。   (function() {  console.log("IIFE called");  })();   * **用途**：   + 只执行一次的初始化代码   + 创建局部作用域，防止变量污染全局作用域   **2. 作用域**   * **全局作用域**：函数外声明的变量，全局可访问。 * **局部作用域**：函数内部声明的变量，仅在函数内可访问。   **3. 作用域链**   * **定义**：嵌套函数在当前作用域找不到变量时，会沿着作用域链向上查找直至全局作用域。   var a = 100;  function outer() {  var b = 200;  function inner() {  var c = 300;  console.log(a); // 100  console.log(b); // 200  console.log(c); // 300  }  inner();  }  outer();  **4. 闭包**   * **定义**：函数能够记住并访问其词法作用域，即使函数在其词法作用域之外执行。   function outer() {  var a = 100;  return function inner() {  console.log(a); // 100  }  }  var fn = outer();  fn();  **5. 经典面试题 - IIFE 和作用域**   * **IIFE 用于循环中的变量隔离**：   for (var i = 0; i < 5; i++) {  (function(j) {  console.log(j); // 输出：0, 1, 2, 3, 4  })(i);  }   * **解释**：IIFE 创建了一个新的作用域，使得每次循环的 i 值作为参数传递给 IIFE，从而避免变量提升导致的错误。   日期2024.7.25 |