Day26笔记

|  |
| --- |
| JavaScript中的原型与对象  一、JavaScript 原型与原型链  1.1 函数  function add(a, b) {  return a + b;  }  function Person(name, age) {  this.name = name;  this.age = age;  }  var p1 = new Person('张三', 18);  函数不仅是基本的操作单元，它还可以作为构造函数来创建对象。理解函数的双重角色（普通函数和构造函数）对于掌握原型链至关重要。`new` 操作符在创建对象时，背后会涉及到原型链的构建。  1.2 属性  \_\_proto\_\_: 对象的私有属性，指向其原型对象。  prototype: 函数独有属性，指向实例对象的原型对象。  constructor: 对象的构造函数属性，指向创建该对象的函数。  console.log(p1.\_\_proto\_\_ === Person.prototype); // true  console.log(p1.constructor); // Person  这些属性揭示了对象与其原型对象之间的关系。`\_\_proto\_\_` 和 `prototype` 在继承和属性查找过程中起着核心作用，而 `constructor` 则指明了对象的来源。  1.2.1 prototype  Person.prototype.sayHi = function() {  console.log(this.name, '说hi');  };  p1.sayHi(); // 张三 说hi  p2.sayHi(); // 李四 说hi  `prototype` 是实现对象共享属性和方法的关键。将方法定义在 `prototype` 上可以让所有实例共享该方法，避免了重复定义。但需要注意的是，原型方法的共享性意味着修改时需要考虑所有实例的影响。  二、深入理解对象  2.1 对象的创建  function createStudent(name, age) {  var stu = new Object();  stu.name = name;  stu.age = age;  stu.sayHi = function() {  console.log(this.name + '说Hi');  };  return stu;  }  var stu1 = createStudent('张三', 18);  ```  手动创建对象虽然简单，但在大量创建时效率低且容易出错。构造函数或工厂模式封装了对象的创建逻辑，可以大大提高代码的复用性和维护性。  2.2 instanceof 操作符  console.log(stu1 instanceof Student); // true  `instanceof` 检查对象是否是某个构造函数的实例，其本质是通过原型链判断构造函数的 `prototype` 是否出现在对象的原型链上。这个操作符在类型检查和多态性实现中非常重要。  2.3 组合模式  Student.prototype.sayHi = function() {  console.log(this.name + '说Hi');  };  var stu1 = new Student('张三', 18);  var stu2 = new Student('李四', 20);  console.log(stu1.sayHi === stu2.sayHi); // true  将方法定义在原型上是提高性能和代码可维护性的常见策略，避免了每个实例重复创建方法。但要注意的是，所有实例共享同一个方法，如果需要特定实例有不同的行为，需要对方法进行适当扩展或覆盖。  2.4 in 操作符  console.log('name' in stu1); // true  `in` 操作符不仅检查对象自身是否有某个属性，还会沿着原型链查找。这在调试和验证对象属性时非常有用，但也需要注意避免意外地访问到继承自原型的属性。  总结：  在 JavaScript 中，原型链和对象的关系决定了属性和方法的查找规则。理解 `\_\_proto\_\_`、`prototype` 和 `constructor` 等属性，以及如何使用 `instanceof`、`in` 等操作符来操控对象，能够帮助我们更深入地掌握 JavaScript 的面向对象编程思想。这些知识点不仅在原生 JS 中重要，在使用如 Vue 这样依赖原型机制的框架时也尤为关键。  日期2024.8.20 |