36. 如何模拟类的 private 属性?

📓 技术笔记：如何模拟类的 private 属性

1. 核心概念 (Core Concept)

在 JavaScript 早期的版本中，并没有内置的、像其他面向对象语言那样严格的 `private` 关键字来限制属性和方法的访问。因此，“模拟 private 属性”是指利用 JavaScript 语言自身的特性（如闭包、命名约定等）来实现对类（或构造函数）内部数据的封装，使其在对象实例外部不可直接访问。

2. 为什么需要它？ (The "Why")

1. **数据封装 (Data Encapsulation):** 私有属性有助于隐藏内部实现细节，只暴露必要的公共接口，这是面向对象设计的重要原则之一，可以提高代码的可维护性和安全性。
2. **避免意外修改 (Prevent Accidental Modification):** 防止外部代码随意修改对象的内部状态，确保对象行为的可控性。
3. **清晰的接口 (Clear Interface):** 明确区分哪些是供外部使用的公共部分，哪些是内部实现细节，使类的使用者更容易理解如何正确地使用它。

3. API 与用法 (API & Usage)

由于 JavaScript 在 ES2020 之前没有原生的私有属性语法，主要的模拟方法依赖于闭包（Closure）或 ES2020 引入的 `#` 私有字段语法。

方法 1: 使用闭包模拟 private 属性 (传统方法)

这是在 ES6 Class 语法广泛使用之前，最常见且有效的方法。它通过在一个函数作用域（通常是构造函数内部）定义变量来实现“私有性”，这些变量只能在该函数内部或其内部定义的、能够访问这些变量的函数（即构成闭包）中访问。

```
function Counter() {
  // 私有变量，只能在此函数内部访问
  let count = 0;

  // 公共方法，可以访问私有变量 count
  this.increment = function() {
    count++;
    console.log(count);
  };

  this.getCount = function() {
    return count;
  };
```

```
}

const myCounter = new Counter();

myCounter.increment(); // 输出: 1
myCounter.increment(); // 输出: 2
console.log(myCounter.getCount()); // 输出: 2

// 外部无法直接访问 count 变量
// console.log(myCounter.count); // indefind
```

方法 2: 使用 `#` 私有字段 (ES2020 新语法)

ES2020 引入了 `private class fields` 提案，允许在类中使用 `#` 前缀来定义真正的私有属性和方法，这是目前官方推荐、最符合“私有”语义的做法。

```
class Developer {
  // 私有属性
  #salary;
  #privateMethod() {
      console.log("This is a private method.");
  }

  constructor(name, salary) {
    this.name = name;
    this.#salary = salary; // 在类内部可以访问私有属性
  }

  getSalary() {
    // 在类内部可以通过 # 访问私有属性
    return this.#salary;
  }

  callPrivateMethod() {
      this.#privateMethod(); // 在类内部可以调用私有方法
  }
}

const dev = new Developer("Alice", 100000);

console.log(dev.name); // 输出: Alice
console.log(dev.getSalary()); // 输出: 100000

// 外部无法直接访问或调用私有属性/方法
// console.log(dev.#salary); // SyntaxError
// dev.#privateMethod(); // SyntaxError
dev.callPrivateMethod(); // 输出: This is a private method.
```

4. 关键注意事项 (Key Considerations)

- **闭包方法的“私有”是模拟：** 使用闭包实现的私有性并非语法层面的强制限制，而是通过作用域链实现的。私有变量并不存在于实例对象本身上，也无法通过 `for...in` 或 `Object.keys()` 等方式枚举。
- **命名约定 (Convention):** 在 ES2020 `#` 语法未普及之前，开发者常使用下划线前缀（如 `_privateProperty`）来表示该属性 intended 为私有，但这只是一种约定，外部代码仍然可以访问和修改。这种方法不具备真正的私有性质。
- **性能开销 (Closure):** 使用闭包模拟私有属性时，每个实例都会创建一个新的闭包作用域和其中定义的函数，这相比于原型链上的公共方法，可能会带来轻微的内存和性能开销（尤其是在创建大量对象实例时）。
- **ES2020 `#` 的兼容性：** 使用 `#` 私有字段是未来的趋势，但需要注意其浏览器和 Node.js 环境的兼容性。现代开发通常会通过 Babel 等工具进行转译，以确保在旧环境中的运行。

5. 参考资料 (References)

- **MDN Web Docs - Closures:** https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures (解释了闭包如何实现私有性)
- **MDN Web Docs - Public and private class fields:** https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/Public_class_fields#private_class_fields (ES2020 `#` 私有字段的官方文档)
- **Exploring JS - Private class fields:** https://exploringjs.com/es6/ch_classes.html#sec_private-instance-fields (一本广受认可的 JavaScript 书籍对私有字段的介绍)