31. 手写 new 的实现逻辑

1. 核心概念 (Core Concept)

`new` 操作符是一个内置操作符，用于创建一个用户定义的对象类型的实例，或者具有构造函数的内置对象的实例。手写 `new` 的实现逻辑，是指模拟 `new` 操作符在底层执行的一系列步骤，以便更好地理解其工作原理。

2. 为什么需要它？ (The "Why")

1. **深入理解原型与继承:** 手写 `new` 迫使开发者思考 JavaScript 中的原型链如何构建，以及实例如何继承构造函数的属性和方法。
2. **理解构造函数行为:** 揭示了构造函数在被 `new` 调用时与普通函数调用时的区别（如 `this` 的指向）。
3. **面试考察点:** 这是前端面试中常考的题目，用于考察开发者对 JavaScript 基础概念的掌握深度。

3. API 与用法 (API & Usage)

`new` 操作符本身没有公共的 API 或参数列表，它是一个语法结构。其标准行为（根据 ECMA-262 规范）可以概括为以下几个核心步骤：

1. **创建新对象 (Create New Object):** 创建一个全新的、空的普通对象 (`{}`).
2. **设置原型链 (Set Prototype):** 将这个新创建的对象的内部原型 (`[[Prototype]]`) 设置为构造函数 `Constructor.prototype` 的值。
3. **绑定 `this` 并执行构造函数 (Bind `this` & Execute Constructor):** 将这个新对象作为构造函数调用时的 `this` 值。执行构造函数体。
4. **返回值处理 (Return Value Handling):**
   - 如果构造函数返回了一个**非原始值**（对象、数组、函数等），则返回这个返回值。
   - 如果构造函数返回了一个**原始值**（字符串、数字、布尔、null、undefined、Symbol、BigInt）或者没有 `return` 语句，则返回第一步创建的新对象。

以下是一个手写模拟 `new` 的函数示例，它试图模拟上述行为：

```
/**
 * 模拟实现 new 操作符的函数
 * @param {Function} Constructor - 构造函数
 * @param {...any} args - 传递给构造函数的参数
 * @returns {object} - 新创建的对象实例
 */
function myNew(Constructor, ...args) {
  // 1. 创建一个全新的、空的普通对象
  const obj = {};
```

```
  // 2. 将这个新创建的对象的内部原型设置为构造函数 Constructor.prototype 的值
  // 推荐使用 Object.create() 来创建对象并设置原型
  // const obj = Object.create(Constructor.prototype);

  // 或者传统方式，但不推荐直接修改 [[Prototype]]
  // obj.__proto__ = Constructor.prototype;

  // 为了模拟原生的行为，并考虑到 Object.create 的意图与步骤1&2更贴合，我们直接用
Object.create
  // 如果严格按照规范第一步先创建空对象再关联原型，代码会有微小差异但核心思想一致
  // 这里选择更接近最终状态的 Object.create 方式，简洁且常用
  const instance = Object.create(Constructor.prototype);


  // 3. 将这个新对象作为构造函数调用时的 `this` 值，并执行构造函数体
  const result = Constructor.apply(instance, args);

  // 4. 处理返回值：
  // 如果构造函数返回了一个非原始值，则返回这个返回值。
  // 否则（返回原始值或无返回值），返回第一步创建的新对象（即 instance）。
  const isObject = typeof result === 'object' && result !== null;
  const isFunction = typeof result === 'function';

  if (isObject || isFunction) {
    return result;
  }

  return instance;
}

// 示例用法：
function Person(name, age) {
  this.name = name;
  this.age = age;
  // return 123; // 返回原始值，会忽略
  // return { company: 'Baidu' }; // 返回对象，会优先返回此对象
}

Person.prototype.sayHello = function() {
  console.log(`Hello, my name is ${this.name} and I am ${this.age} years
old.`);
};

const person1 = new Person('Alice', 30);
const person2 = myNew(Person, 'Bob', 25);

console.log(person1); // Person { name: 'Alice', age: 30 }
console.log(person2); // Person { name: 'Bob', age: 25 }
person1.sayHello(); // Hello, my name is Alice and I am 30 years old.
```

```
person2.sayHello(); // Hello, my name is Bob and I am 25 years old.

// 验证返回对象的情况
function AnotherPerson(name) {
    this.name = name;
    return { greeting: 'Hi!' };
}

const person3 = new AnotherPerson('Charlie');
const person4 = myNew(AnotherPerson, 'David');

console.log(person3); // { greeting: 'Hi!' }
console.log(person4); // { greeting: 'Hi!' }

// 验证返回原始值的情况
function YetAnotherPerson(name) {
    this.name = name;
    return 123;
}

const person5 = new YetAnotherPerson('Eve');
const person6 = myNew(YetAnotherPerson, 'Frank');

console.log(person5); // YetAnotherPerson { name: 'Eve' }
console.log(person6); // YetAnotherPerson { name: 'Frank' }
```

4. 关键注意事项 (Key Considerations)

1. **原型设置方式:** 模拟原型链的关键在于设置新对象的 `[[Prototype]]` 指向构造函数的 `prototype` 属性。`Object.create()` 是一个标准且推荐的方式来同时创建对象并指定其原型。直接修改 `__proto__` 虽然在许多环境中有效，但并非所有环境都支持，且性能和规范上都有争议。
2. **`this` 绑定:** 构造函数执行时的 `this` 必须绑定到新创建的对象上。这通常通过 `call()` 或 `apply()` 方法实现。
3. **返回值判断:** 必须正确处理构造函数返回值的逻辑。这是模拟 `new` 操作符行为中一个重要的细节。如果构造函数显式返回了一个非原始值，那么 `new` 的结果就是这个返回值，而不是新创建的对象。
4. **参数传递:** `new` 操作符调用构造函数时可以传递参数，模拟函数也需要能接收并传递这些参数，这通常通过 `apply()` 或 `call()` 与拓展运算符 (`...args`) 结合来实现。

5. 参考资料 (References)

- **MDN Web Docs - new 操作符:** https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/new

- **ECMA-262 规范 (Current Draft) - 12.3.3.1 Runtime Semantics: EvaluateNew:** （此链接指向当前草案，具体章节号可能随规范版本变化，核心逻辑一致，需要查阅对应版本的 ES 规范）https://tc39.es/ecma262/ (搜索 "EvaluateNew")
- **MDN Web Docs - Object.create():** https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Object/create
- **MDN Web Docs - Function.prototype.apply():** https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Function/apply
- **MDN Web Docs - Function.prototype.call():** https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Function/call