2.数据类型和 typeof 的陷阱

1. 核心概念 (Core Concept)

JavaScript 的数据类型分为两大类：**原始类型 (Primitive Types)** 和**对象类型 (Object Types)**。`typeof` 是一个一元操作符，用于返回一个表示未经计算的操作数类型的字符串。

- **7 种原始类型**: `string`, `number`, `boolean`, `null`, `undefined`, `symbol` (ES6), `bigint` (ES2020)。
- **1 种对象类型**: `object` (包括 `Array`, `Function`, `Date`, `RegExp` 等所有非原始类型)。

2. 为什么需要理解 `typeof` 的陷阱？ (The "Why")

在进行类型判断时，`typeof` 是最常用的工具之一，但它并非万能，存在一些历史遗留的、与直觉相悖的行为。理解这些陷阱可以：

1. **避免错误的类型判断**: 尤其是在处理 `null`、数组和对象时，`typeof` 的结果可能导致逻辑错误。
2. **编写更健壮的代码**: 了解其局限性，才能选择更精确的类型检测方法（如 `Object.prototype.toString.call()`）来处理复杂场景。
3. **深入理解 JS 内部机制**: `typeof null` 的问题源于 JavaScript 早期的实现 bug，了解它有助于理解语言的演变。

3. API 与用法 (API & Usage)

`typeof` 操作符后跟一个操作数，返回其类型的字符串表示。

经典用法示例

```
// 原始类型
console.log(typeof 'hello');      // "string"
console.log(typeof 123);          // "number"
console.log(typeof true);         // "boolean"
console.log(typeof Symbol('id')); // "symbol"
console.log(typeof 123n);         // "bigint"
console.log(typeof undefined);    // "undefined"

// 对象类型
console.log(typeof { a: 1 });     // "object"
console.log(typeof [1, 2, 3]);    // "object" (陷阱)
console.log(typeof new Date());   // "object"
```

```
// 函数
console.log(typeof function() {});// "function" (特殊的对象)
```

`typeof` 的主要陷阱

陷阱 1: `typeof null` 返回 `"object"`

这是 JavaScript 历史遗留的一个 bug。在 JavaScript 最初的实现中，值在底层是以类型标签和值的形式存储的。对象的类型标签是 `0`，而 `null` 在底层被表示为 `NULL` 指针 (大多数平台为 `0x00`)，因此 `typeof null` 错误地返回了 `"object"`。

```
console.log(typeof null); // "object"
```

陷阱 2: 无法区分对象和数组

`typeof` 无法区分 `Array`、`Object` 等具体的对象子类型，除了 `Function`。

```
console.log(typeof []);       // "object"
console.log(typeof {});       // "object"
```

4. 关键注意事项 (Key Considerations)

1. **`null` 的判断**: 判断一个值是否为 `null`，最可靠的方法是使用严格相等 `===`。

```
const myVar = null;
if (myVar === null) {
  console.log('It is null!');
}
```

2. **数组的判断**: 要准确判断一个值是否为数组，应使用 `Array.isArray()`。

```
const arr = [];
console.log(Array.isArray(arr)); // true
console.log(typeof arr);         // "object"
```

3. **精确的类型检测**: 对于需要精确区分各种对象子类型的场景，`Object.prototype.toString.call()` 是最通用的解决方案。它能返回 `"[object Type]"` 格式的字符串，其中 `Type` 是具体的类型。

```
Object.prototype.toString.call('');      // "[object String]"
Object.prototype.toString.call(1);       // "[object Number]"
Object.prototype.toString.call(null);    // "[object Null]"
```

```
Object.prototype.toString.call([]);      // "[object Array]"
Object.prototype.toString.call({});      // "[object Object]"
```

4. **`typeof` 对未声明变量不报错**: 对一个未声明的变量使用 `typeof` 不会抛出 `ReferenceError`，而是返回 `"undefined"`。这在某些场景下可用于安全地检查变量是否存在。

```
// someUndeclaredVar is not declared
console.log(typeof someUndeclaredVar); // "undefined"
// console.log(someUndeclaredVar); // ReferenceError
```

5. 参考资料 (References)

- MDN Web Docs: typeof
- MDN Web Docs: JavaScript data types and data structures
- ECMAScript® 2025 Language Specification: The typeof Operator