# 6.对象遍历方式对比（for in、Object.keys、Reflect）

## 1. 核心概念 (Core Concept)

在 JavaScript 中，遍历对象属性有多种方式，主要区别在于它们处理**可枚举性 (enumerable)**、**原型链 (prototype chain)** 属性和 **Symbol** 属性的能力。

- `for...in`：遍历对象自身及其原型链上所有**可枚举**的字符串属性。
- `Object.keys()`：返回一个数组，包含对象自身所有**可枚举**的字符串属性名。
- `Object.getOwnPropertyNames()`：返回一个数组，包含对象自身所有字符串属性名（无论是否可枚举）。
- `Object.getOwnPropertySymbols()`：返回一个数组，包含对象自身所有 Symbol 属性名。
- `Reflect.ownKeys()`：返回一个数组，包含对象自身的所有属性名（包括字符串和 Symbol，无论是否可枚举）。

## 2. 为什么需要对比它们？ (The "Why")

1. **选择正确的工具**: 不同的遍历需求需要不同的工具。例如，是只需要访问对象自身的属性，还是也需要原型链上的？是只关心可枚举属性，还是所有属性？是否需要处理 Symbol？错误的选择可能导致 bug 或性能问题。
2. **理解属性描述符**: 这些遍历方式的行为差异根植于 JavaScript 对象的属性描述符（`enumerable`，`configurable`，`writable`，`value`）。对比它们有助于加深对对象底层机制的理解。
3. **编写健壮的库和框架**: 在编写通用库或框架时，需要处理各种来源和类型的对象。`Reflect.ownKeys()` 提供了最全面的属性遍历方式，是实现复制、代理、序列化等功能的坚实基础。

## 3. API 与用法 (API & Usage)

**准备一个复杂的对象用于演示:**

```
const sym = Symbol('demo');

const proto = {
  protoProp: 'proto value'
};

const obj = Object.create(proto);
obj.ownProp = 'own value';
obj[sym] = 'symbol value';
```

```
Object.defineProperty(obj, 'nonEnumProp', {
  value: 'non-enumerable value',
  enumerable: false
});
```

## `for...in`

遍历自身及原型链上的可枚举字符串属性。

```
const result = [];
for (let key in obj) {
  result.push(key);
}
// 关键：通常与 hasOwnProperty 配合使用，以过滤掉原型链属性
const ownResult = [];
for (let key in obj) {
  if (Object.prototype.hasOwnProperty.call(obj, key)) {
    ownResult.push(key);
  }
}

console.log(result);    // ["ownProp", "protoProp"] (包含了原型链上的属性)
console.log(ownResult); // ["ownProp"]
```

## `Object.keys()`

只返回自身的可枚举字符串属性。

```
console.log(Object.keys(obj)); // ["ownProp"]
```

## `Reflect.ownKeys()`

返回自身的所有属性（字符串、Symbol、可枚举、不可枚举）。

```
console.log(Reflect.ownKeys(obj)); // ["ownProp", "nonEnumProp",
Symbol(demo)]
```

# 综合对比表格

| 方法 | 遍历自身属性? | 遍历原型链属性? | 遍历可枚举属性? | 遍历 Symbol 属性? |
|---|---|---|---|---|
| `for...in` | ✅ | ✅ | ✅ | ❌ |
| `Object.keys()` | ✅ | ❌ | ✅ | ❌ |

| 方法 | 遍历自身属性? | 遍历原型链属性? | 遍历可枚举属性? | 遍历 Symbol 属性? |
| --- | --- | --- | --- | --- |
| `Object.getOwnPropertyNames()` | ✅ | ❌ | ✅ & ❌ | ❌ |
| `Object.getOwnPropertySymbols()` | ✅ | ❌ | N/A | ✅ |
| `Reflect.ownKeys()` | ✅ | ❌ | ✅ & ❌ | ✅ |

# 4. 关键注意事项 (Key Considerations)

1. **`for...in` 的陷阱**: 默认情况下， `for...in` 会遍历原型链。这是一个常见的问题源头。为了避免意外行为，**必须**使用 `hasOwnProperty()` 来过滤掉原型链上的属性。
2. **`Reflect.ownKeys()` 的全面性**: `Reflect.ownKeys()` 提供了最完整的对象自身属性列表，等价于 `Object.getOwnPropertyNames(obj).concat(Object.getOwnPropertySymbols(obj))` 。在需要完整复制或代理一个对象时，这是最可靠的选择。
3. **遍历顺序**:
   - 对于字符串键，所有方法都遵循一个大致的顺序：先遍历所有整数索引键（按升序），然后遍历所有其他字符串键（按插入顺序）。
   - `Reflect.ownKeys` 和 `Object.getOwnPropertyNames` 等会先返回所有字符串键，然后是所有 Symbol 键。
4. **性能**: `Object.keys` 通常比 `for...in` + `hasOwnProperty` 更快，因为它不涉及原型链查找。 `Reflect.ownKeys` 因为需要处理更多类型的属性，开销可能略高，但提供了最全面的结果。

# 5. 参考资料 (References)

- MDN Web Docs: for...in
- MDN Web Docs: Object.keys()
- MDN Web Docs: Reflect.ownKeys()
- MDN Web Docs: Enumerability and ownership of properties