Cơ chế dọn dẹp bộ nhớ (Garbage Collection) trong JavaScript

JavaScript có **cơ chế tự động quản lý bộ nhớ** bằng **Garbage Collection (GC)**, giúp **giải phóng bộ nhớ không còn được sử dụng** để tối ưu hiệu suất.

---

1. Cách JavaScript quản lý bộ nhớ

Trong JavaScript, bộ nhớ được cấp phát theo các bước sau:

1. **Cấp phát bộ nhớ** khi khai báo biến, đối tượng, mảng, hàm...
2. **Sử dụng bộ nhớ** bằng cách thực hiện thao tác trên dữ liệu.
3. **Thu hồi bộ nhớ không còn sử dụng** (Garbage Collection).

---

2. Cơ chế Garbage Collection hoạt động như thế nào?

JavaScript sử dụng phương pháp **"Mark and Sweep" (Đánh dấu và Quét)** để xác định và giải phóng bộ nhớ không còn tham chiếu.

🔷 Bước 1: Đánh dấu (Mark)

- Trình thu gom rác (GC) duyệt qua tất cả các biến, đối tượng, và kiểm tra xem chúng có thể truy cập được không.
- Nếu một biến **vẫn còn được tham chiếu**, nó sẽ được đánh dấu là **vẫn còn sử dụng**.
- Nếu một biến **không còn được tham chiếu**, nó sẽ được đánh dấu là **có thể thu hồi**.

🔷 Bước 2: Quét và dọn dẹp (Sweep)

- Các biến không còn tham chiếu đến (đã bị đánh dấu là không sử dụng) sẽ bị thu hồi.
- Vùng nhớ của chúng được giải phóng để sử dụng cho dữ liệu mới.

Ví dụ minh họa cơ chế GC trong JavaScript

```
function createUser() {
    let user = { name: "Khang" }; // Cấp phát bộ nhớ cho object
    return user;
}

let person = createUser(); // Object vẫn có tham chiếu (person)
person = null; // Không còn tham chiếu -> sẽ bị GC dọn dẹp
```

🛠️ **Giải thích:**

- Khi `person = null`, object `{ name: "Khang" }` **không còn tham chiếu** và sẽ bị thu hồi bởi Garbage Collector.

3. Khi nào bộ nhớ bị thu hồi?

- **Biến toàn cục (Global variables)**: Không bị thu hồi trong suốt vòng đời chương trình, trừ khi trình duyệt đóng.
- **Biến cục bộ (Local variables)**: Khi hàm kết thúc và biến không còn được tham chiếu, nó sẽ bị thu hồi.
- **Objects, Arrays, Functions**: Khi không còn tham chiếu đến chúng, bộ nhớ sẽ được thu hồi.

4. Một số vấn đề về bộ nhớ cần lưu ý

🔶 Vấn đề 1: Memory Leak (Rò rỉ bộ nhớ)

Nếu không quản lý tốt tham chiếu, bộ nhớ có thể bị chiếm dụng lâu dài, gây chậm hệ thống.

Ví dụ: Rò rỉ bộ nhớ do quên xóa event listener

```
function createButton() {
    let button = document.createElement("button");
    button.innerText = "Click me";

    button.addEventListener("click", function () {
        console.log("Clicked!");
    });

    document.body.appendChild(button);
}

createButton(); // Gọi nhiều lần sẽ tạo nhiều button mà không xóa bộ nhớ
```

🛠️ **Giải pháp:** Xóa event listener khi không cần nữa.

```
button.removeEventListener("click", handler);
```

🔶 Vấn đề 2: Closure giữ tham chiếu không cần thiết

```
function outer() {
    let largeArray = new Array(1000000).fill("🚀");

    return function inner() {
        console.log(largeArray.length);
    };
}
```

```
let myFunction = outer();
myFunction(); // Closure giữ lại largeArray trong bộ nhớ
```

🛠️ **Giải pháp:** Gán `null` để xóa tham chiếu nếu không cần nữa.

```
myFunction = null;
```

---

5. Tóm tắt

✅ **JavaScript sử dụng Garbage Collection (GC) để thu hồi bộ nhớ không còn sử dụng.**
✅ **Cơ chế chính: "Mark and Sweep" (Đánh dấu và Quét).**
✅ **Cần chú ý đến Memory Leak do closure, event listener, hoặc object giữ tham chiếu quá lâu.**
✅ **Giải pháp:** Xóa event listener, gán `null`, hoặc tránh giữ tham chiếu không cần thiết.

👉 **Nếu không quản lý bộ nhớ tốt, ứng dụng có thể chạy chậm hoặc gặp lỗi do hết bộ nhớ!** 🚀