메모리 누수(Memory Leaks)

Chrome 개발자 도구로 하는 디버깅

2025.01.26
정상영

1. 메모리 누수

1) 메모리 누수란?

- 부주의 혹은 프로그램의 오류로 인해 사용되지 않는 메모리를 해제하지 못하는 것
- 예를 들어, 어떠한 변수가 100MB의 메모리를 점유할 때, 이 변수가 사용되지 않더라도 메모리에서 해제되지 않고 점유하고 있는 것

1. 메모리 누수

2) 자바스크립트의 메모리 구조

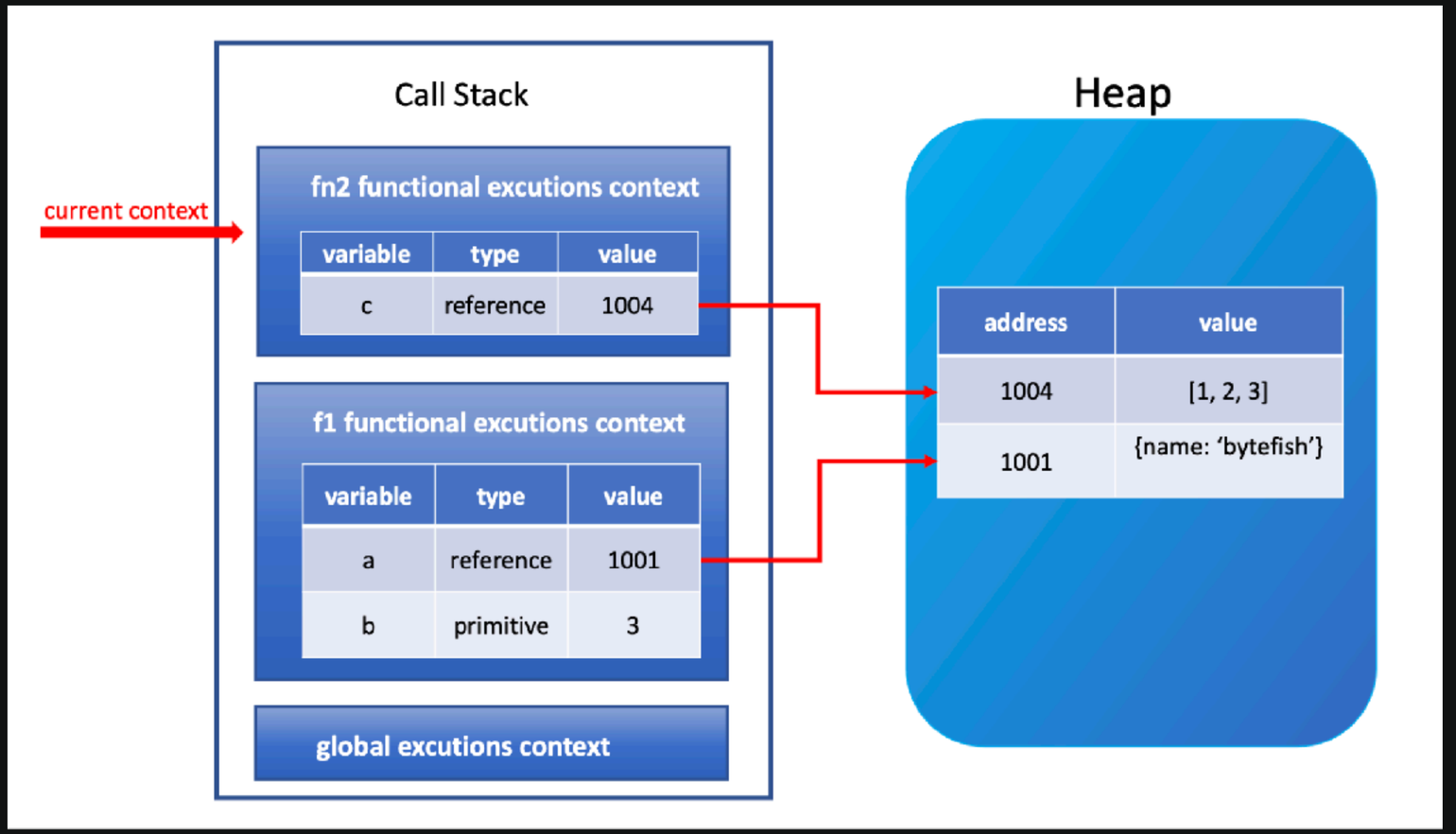
- 자바스크립트 엔진은 콜 스택(Call Stack)과 힙(Heap)을 사용하여 메모리를 관리
- 콜 스택:
  - 실행 컨텍스트를 관리하는 구조
  - 함수 호출 시 콜스택에 함수가 쌓이고, 실행이 끝나면 제거
- 힙: 동적으로 할당된 객체(Object)와 클로저(Closure)와 같은 데이터가 저장되는 구조
  - 사용이 끝난 메모리를 가비지 컬렉터가 회수

1. 메모리 누수

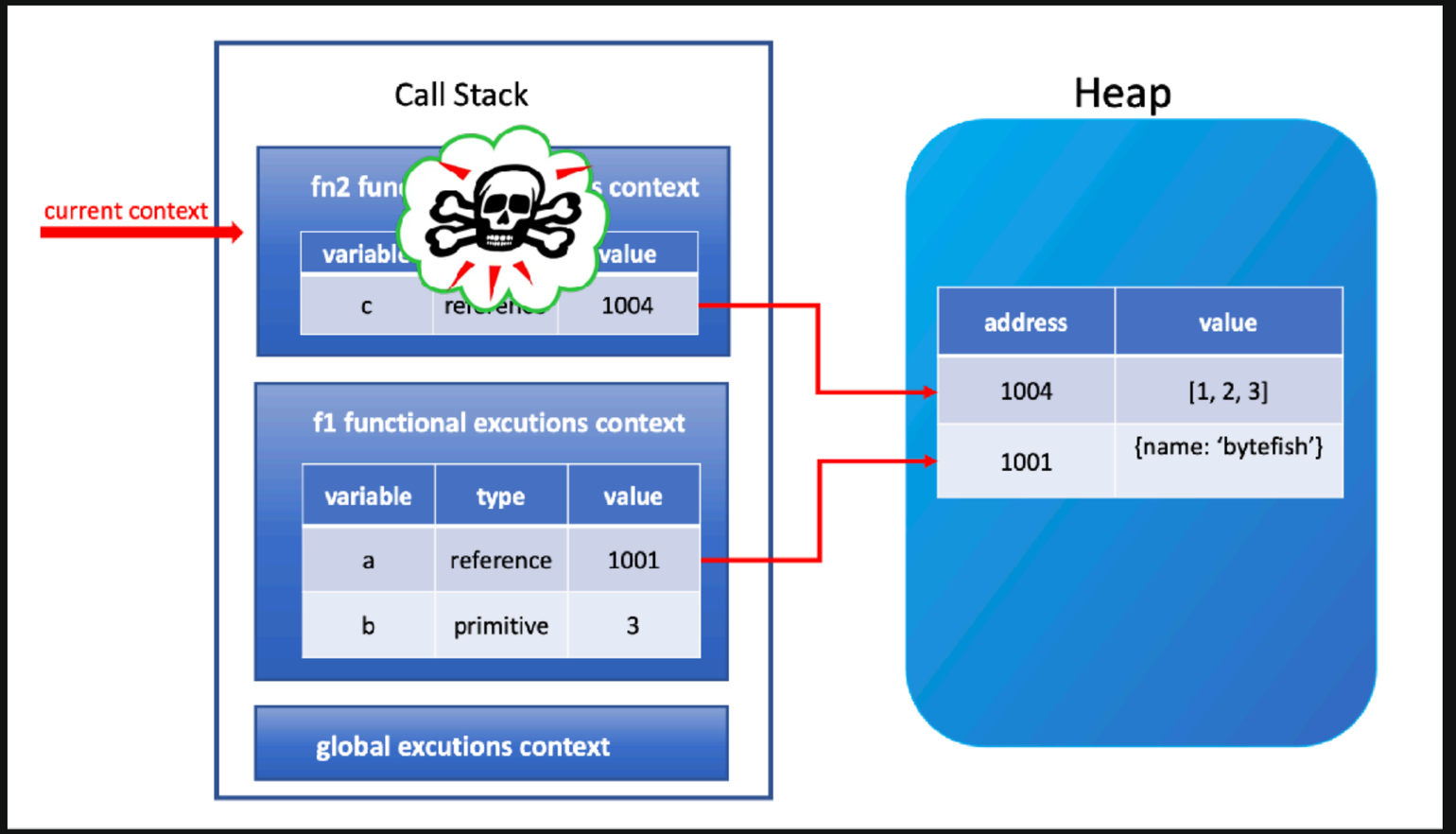
3) 자바스크립트의 가비지 컬렉션

- 가비지 컬렉션: 더는 필요하지 않은 변수 또는 데이터를 정리하는 것
- 자바스크립트는 자동 정리 메커니즘을 사용
- 값이 "도달 가능한지"를 기준으로 작동

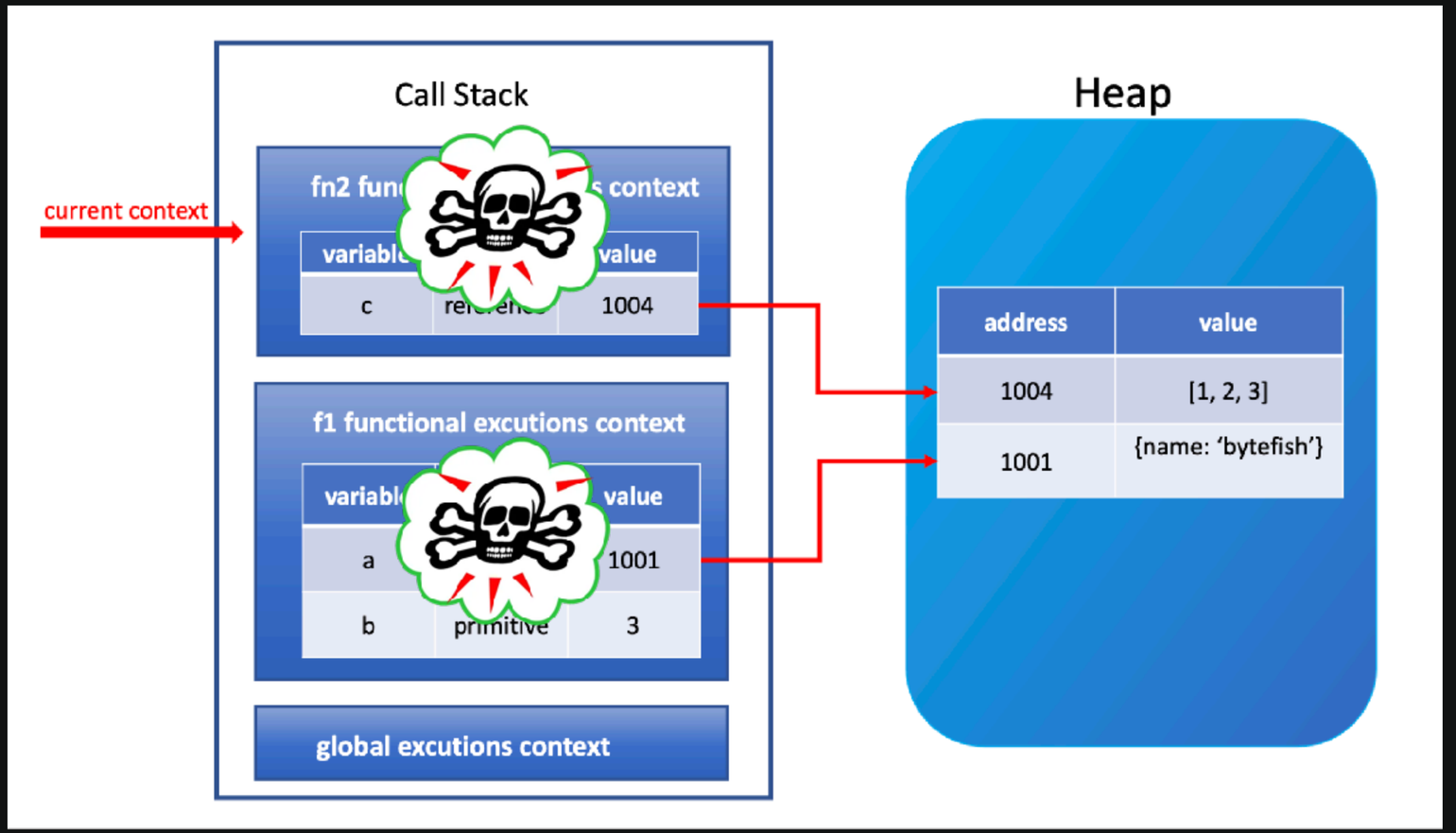
```
function fn1 () {
    let a = {
        name: 'bytefish'
    }

    let b = 3

    function fn2() {
        let c = [1, 2, 3]
    }

    fn2()

    return a
}
let res = fn1()
```



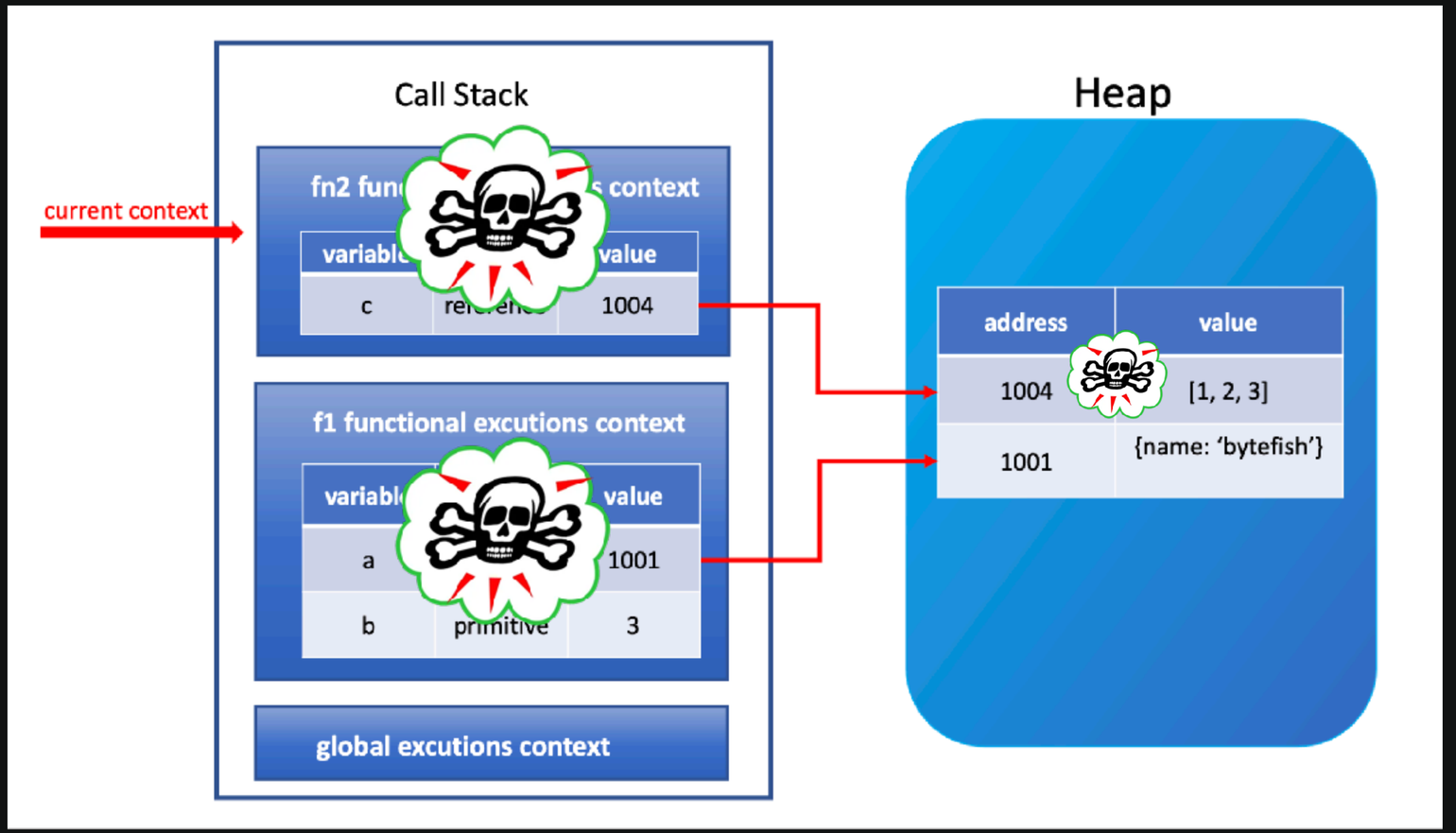
```
function fn1 () {
    let a = {
        name: 'bytefish'
    }

    let b = 3

    function fn2() {
        let c = [1, 2, 3]
    }

    fn2()

    return a
}
let res = fn1()
```



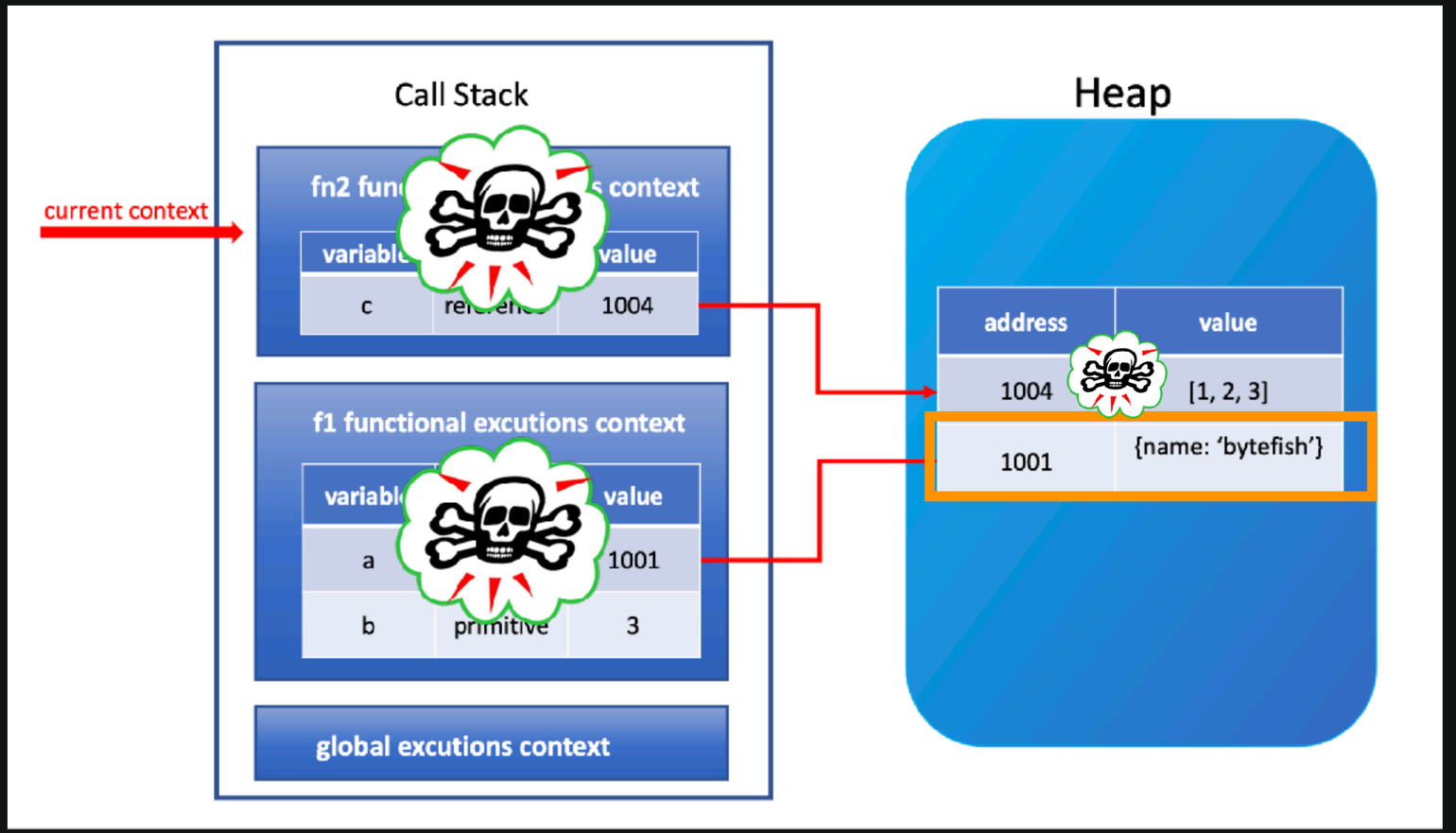
```
function fn1 () {
    let a = {
        name: 'bytefish'
    }

    let b = 3

    function fn2() {
        let c = [1, 2, 3]
    }

    fn2()

    return a
}
let res = fn1()
```



```
function fn1 () {
    let a = {
        name: 'bytefish'
    }

    let b = 3

    function fn2() {
        let c = [1, 2, 3]
    }

    fn2()

    return a
}
let res = fn1()
```

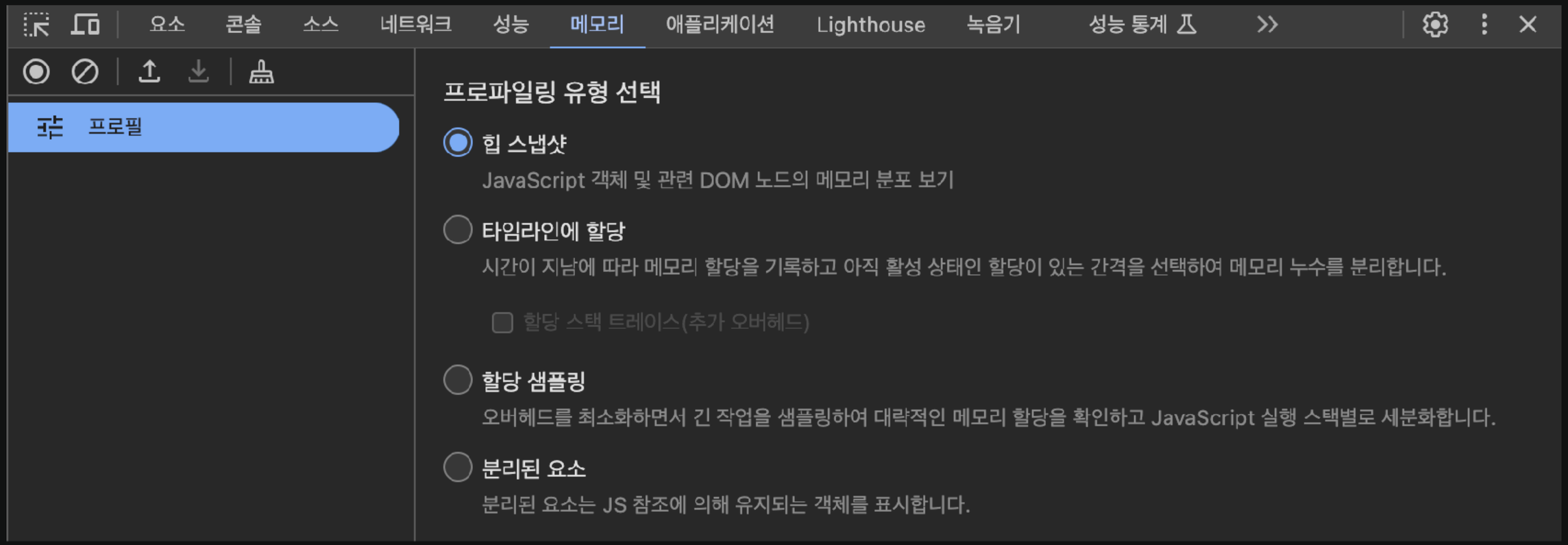


```
function fn1 () {
    let a = {
        name: 'bytefish'
    }

    let b = 3

    function fn2() {
        let c = [1, 2, 3]
    }

    fn2()

    return a
}
let res = fn1()
```

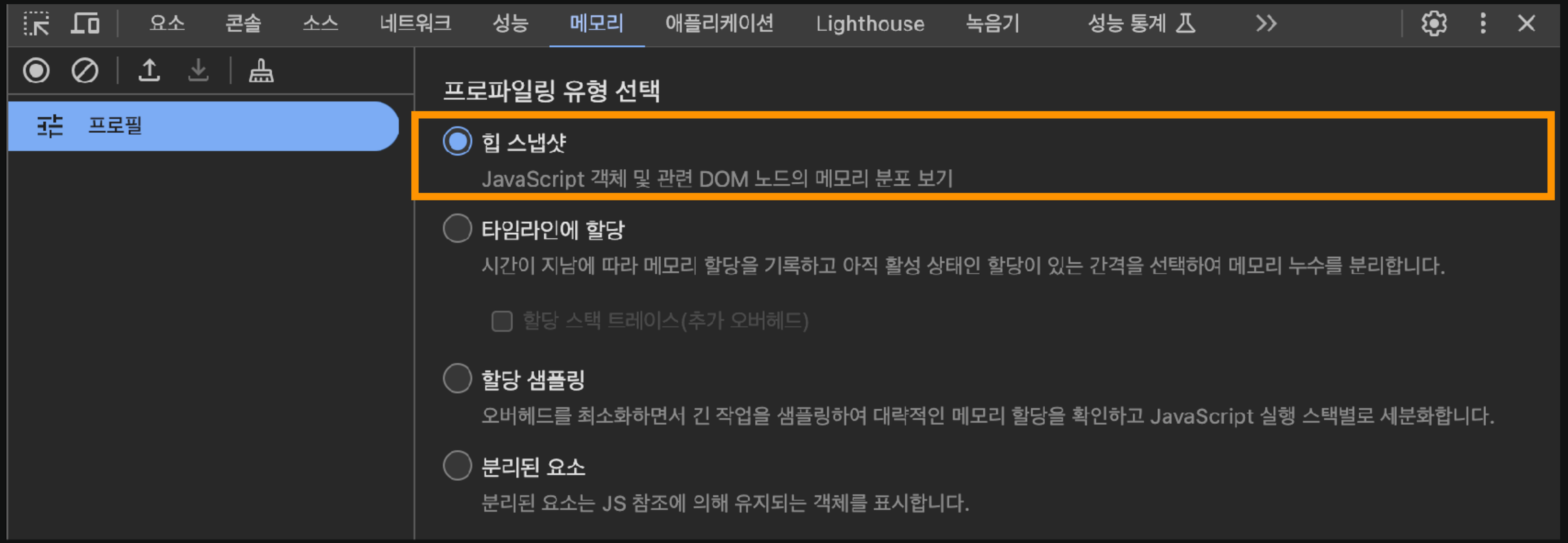


2. 메모리 탭 활용하기

1) 프로파일 선택



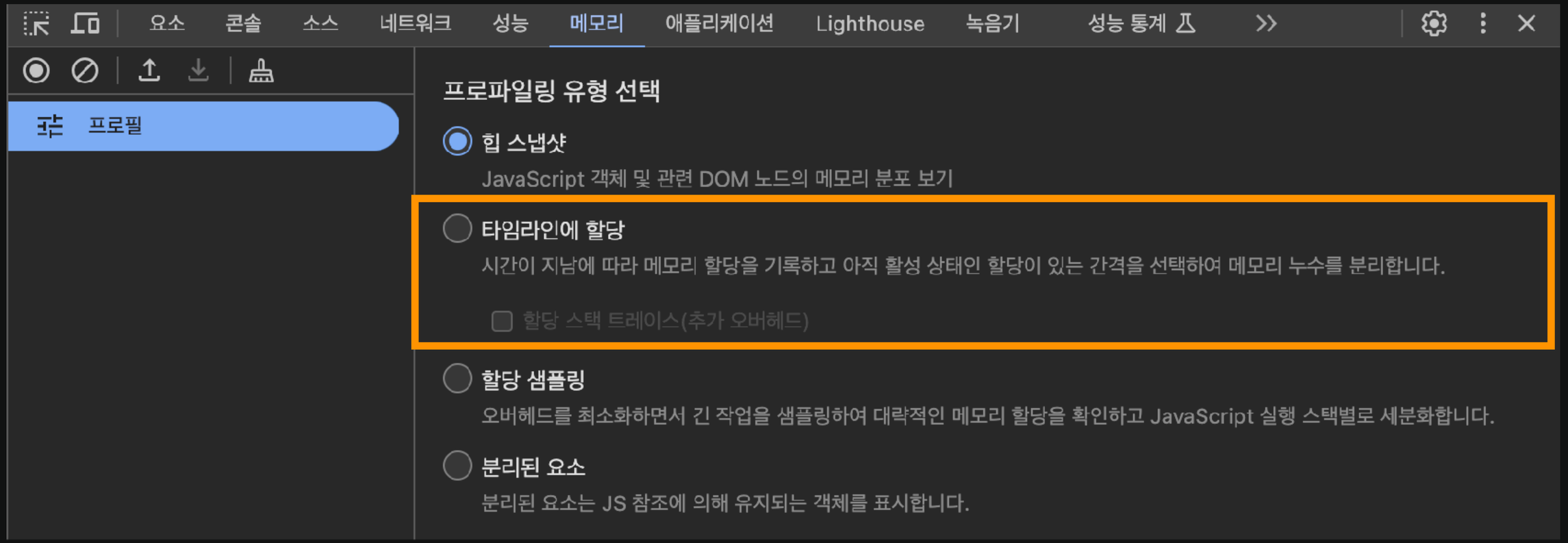
1) 힙 스냅샷


요소
콘솔
소스
네트워크
성능
메모리
애플리케이션
Lighthouse
녹음기
성능 통계
프로필
프로파일링 유형 선택
힙 스냅샷
JavaScript 객체 및 관련 DOM 노드의 메모리 분포 보기
타임라인에 할당
시간이 지남에 따라 메모리 할당을 기록하고 아직 활성 상태인 할당이 있는 간격을 선택하여 메모리 누수를 분리합니다.
할당 스택 트레이스(추가 오버헤드)
할당 샘플링
오버헤드를 최소화하면서 긴 작업을 샘플링하여 대략적인 메모리 할당을 확인하고 JavaScript 실행 스택별로 세분화합니다.
분리된 요소
분리된 요소는 JS 참조에 의해 유지되는 객체를 표시합니다.

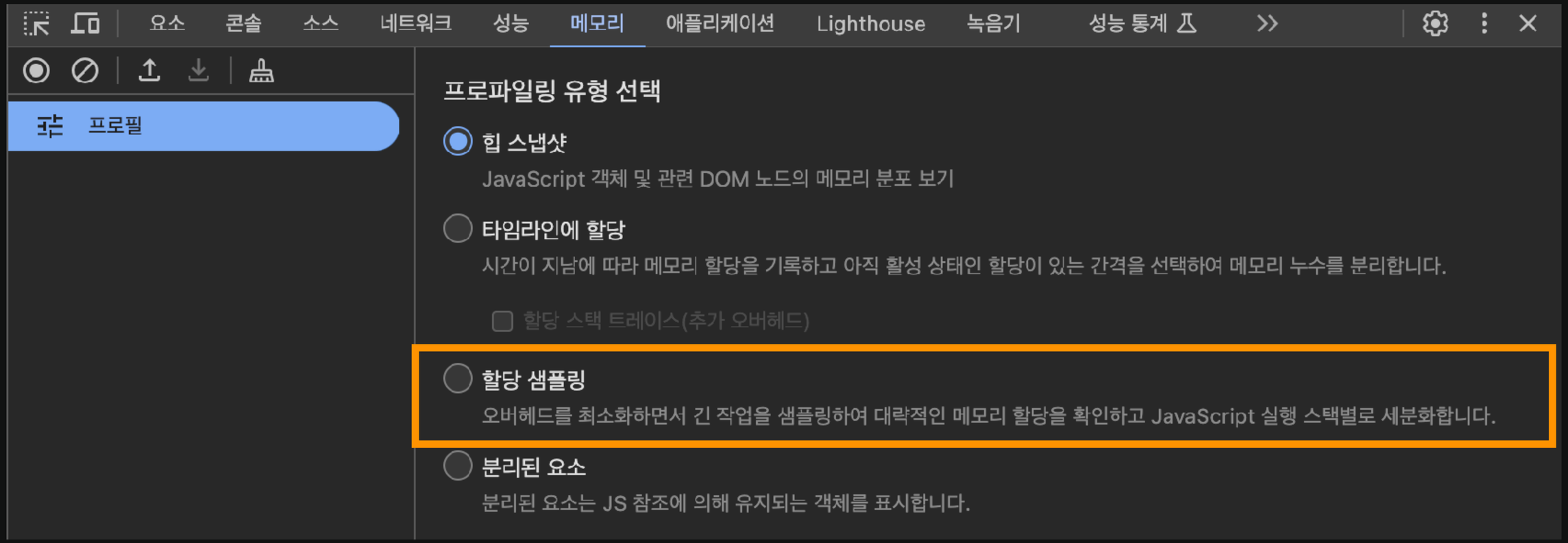
1) 힙 스냅샷

```
const DUMMY_LIST = [];

function App() {
    const handleClick = () => {
        Array.from({ length: 10000000 }).forEach((_, idx) => DUMMY_LIST.push(Math.random() * idx));
        alert('완료!');
    };
    return (
        <>
            <button onClick={handleClick}>Bug Button</button>
        </>
    );
}

export default App;

```

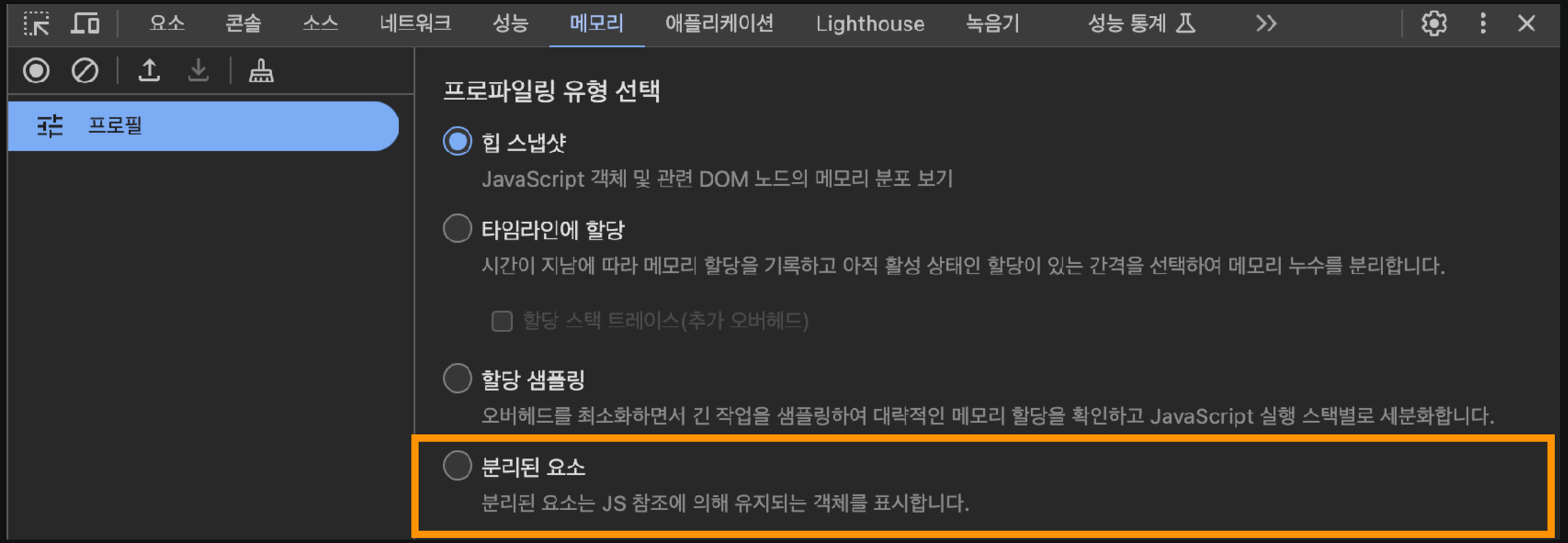
2) 타임라인 할당 계측


요소
콘솔
소스
네트워크
성능
메모리
애플리케이션
Lighthouse
녹음기
성능 통계
프로필
프로파일링 유형 선택
힙 스냅샷
JavaScript 객체 및 관련 DOM 노드의 메모리 분포 보기
타임라인에 할당
시간이 지남에 따라 메모리 할당을 기록하고 아직 활성 상태인 할당이 있는 간격을 선택하여 메모리 누수를 분리합니다.
할당 스택 트레이스(추가 오버헤드)
할당 샘플링
오버헤드를 최소화하면서 긴 작업을 샘플링하여 대략적인 메모리 할당을 확인하고 JavaScript 실행 스택별로 세분화합니다.
분리된 요소
분리된 요소는 JS 참조에 의해 유지되는 객체를 표시합니다.

3) 할당 샘플링


요소
콘솔
소스
네트워크
성능
메모리
애플리케이션
Lighthouse
녹음기
성능 통계
프로필
프로파일링 유형 선택
힙 스냅샷
JavaScript 객체 및 관련 DOM 노드의 메모리 분포 보기
타임라인에 할당
시간이 지남에 따라 메모리 할당을 기록하고 아직 활성 상태인 할당이 있는 간격을 선택하여 메모리 누수를 분리합니다.
할당 스택 트레이스(추가 오버헤드)
할당 샘플링
오버헤드를 최소화하면서 긴 작업을 샘플링하여 대략적인 메모리 할당을 확인하고 JavaScript 실행 스택별로 세분화합니다.
분리된 요소
분리된 요소는 JS 참조에 의해 유지되는 객체를 표시합니다.

4) 분리된 요소


요소
콘솔
소스
네트워크
성능
메모리
애플리케이션
Lighthouse
녹음기
성능 통계
프로필
프로파일링 유형 선택
힙 스냅샷
JavaScript 객체 및 관련 DOM 노드의 메모리 분포 보기
타임라인에 할당
시간이 지남에 따라 메모리 할당을 기록하고 아직 활성 상태인 할당이 있는 간격을 선택하여 메모리 누수를 분리합니다.
할당 스택 트레이스(추가 오버헤드)
할당 샘플링
오버헤드를 최소화하면서 긴 작업을 샘플링하여 대략적인 메모리 할당을 확인하고 JavaScript 실행 스택별로 세분화합니다.
분리된 요소
분리된 요소는 JS 참조에 의해 유지되는 객체를 표시합니다.

3. 메모리 누수 예제

1. 클로저의 잘못된 사용
2. 의도치않게 생성된 전역 변수
3. 제거했지만 여전히 참조되고 있는 DOM 노드
4. console.log()
5. 해제하지 않은 타이머