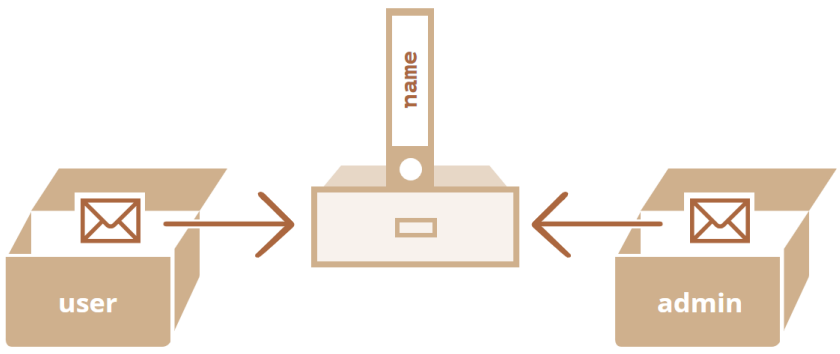
04-2~8. 객체: 기본

참조에 의한 객체 복사

- 원시값은 '**값 그대로**' 저장, 할당되고 복사됨
    - 아래의 예시 코드에서는, 두 개의 독립 변수에 문자열이 저장됨

```
let message = "Hello!";
let phrase = message;
```

- 하지만 객체는 '**참조에 의해(by reference)**' 저장되고 복사됨
    - 즉, 변수에 **객체가 저장된 '메모리 주소'를 참조할 수 있는 값**이 저장됨
    - 이 변수를 복사하면, 참조 값이 복사되고 객체는 복사되지 않음




```
let user = { name: 'John' };
let admin = user; // 객체가 저장된 '메모리 주소' 복사
admin.name = 'Pete'; // 'admin' 참조 값에 의해 변경됨
alert(user.name); // 'Pete'가 출력됨. 'user' 참조 값을 이용해 변경사항을 확인함
```

    - 그렇기에 객체에 접근할 때는 여러 변수 사용 가능
- 객체 비교
    - 그대로 복사해서 비교할 경우, 같은 객체를 참조하기에 일치·동등 비교 모두 참
    - 내용물이 같더라도, 독립된 객체라면 일치·동등 비교 모두 거짓

```
// 참조에 의한 복사
let a = {};
let b = a;
```

```
// 독립된 두 객체
let a = {};
let b = {};
```

```
alert( a == b ); // true
alert( a === b ); // true
```

```
alert( a == b ); // false
alert( a === b ); // 당연히 false
```

객체 복사

- 메모리 주소 복사(defalut)
  - 객체가 할당된 변수 복사 ⇒ **'객체가 저장된 메모리 주소'**를 복사
- 얕은 복사
  - 객체 최상위 레벨만 복사, 중첩된 객체는 참조 값을 복사하기에 같은 값을 참조
  - `for`문, `Object.assign`, `spread 연산자`등으로 가능

```
const original = {
  name: 'John',
  address: {
    city: 'New York',
    zip: '10001'
  }
};

const shallowCopy = { ...original };

shallowCopy.name = 'Jane'; // 독립적으로 변경 가능
shallowCopy.address.city = 'Los Angeles'; // 원본 객체도 영향을 받음

console.log(original.address.city); // 'Los Angeles'
```

- 깊은 복사
  - 객체의 모든 레벨을 복사, 완전히 독립적인 객체
  - 각 값을 검사하면서, 만약 객체라면 객체의 구조도 복사해주는 반복문을 통해 복사
  - Structured cloning algorithm을 사용하거나, `lodash`라이브러리 등으로 가능
- **React의 경우**
  - React는 빠르게 상태 변경 여부를 판단하기 위해 불변성 유지 & 얕은 비교 사용
    - 얕은 비교는 얕은 복사처럼, 최상위 레벨만 비교하는 것
    - React에서 주소값만 비교한다는 말이 많은데(심지어 GPT도 그렇게 말함) 실제로는 주소값 비교 + 모든 Key에 대해서 비교하는 방식으로 동작
    - https://velog.io/@jinhyukoo/React-얕은-비교에-대한-얕은-오해

가비지 컬렉션

- JS도 메모리 관리를 수행함
  - 원시값, 객체, 함수 등 모든 것이 메모리를 차지하기에 이를 정리함
  - 엔진별로 내부 알고리즘이 다르며, 알아서 자동으로 수행함

도달 가능성

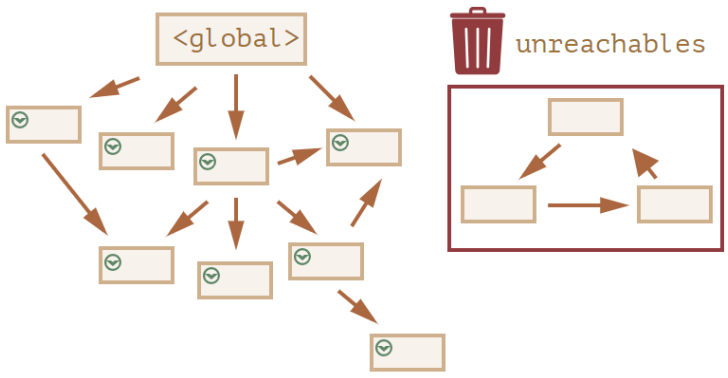
- *도달 가능성(reachability)* 이라는 개념을 사용해 메모리 관리를 수행
  - 도달 가능한 값 → 어떻게든 접근 & 사용이 가능한 값, 두 종류가 있음
    - 단, 외부로 나가는 참조는 도달 가능한 상태에 영향을 주지 않음

1. 루트(root)
   a. 전역 변수
   b. 현재 함수의 지역 변수와 매개변수
   c. 중첩된 함수의 체인에 있는 함수의 변수와 매개변수

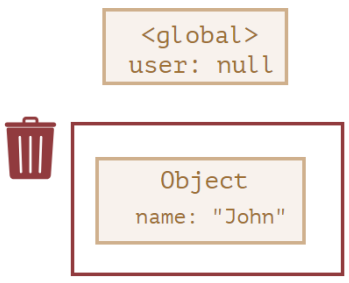
2. 루트에서 참조할 수 있는 값

동작

- 루트에 페인트를 붓는다고 생각하면 쉬움
  1. 루트에서 시작해 모든 참조를 따라가면서 도달 가능한 객체 Mark
  2. 도달할 수 없었던 객체를 삭제




```
let user = {
  name: "John"
};
user = null;
// 가비지 컬렉터가 John 객체 삭제
```



메서드와 this

- **메서드**는 객체의 프로퍼티에 할당된 **함수**
- **this**는 매서드 **내부에서 객체에 접근**하기 위해 사용하는 키워드
  - 외부 변수를 통해 참조할 수도 있지만, 원치 않는 값이 참조될 수 있음

```
let user = {
  name: "John",
  age: 30,
  sayUserName() {
    alert(user.name);
  },
  sayThisName() {
    alert(this.name);
  }
};

let admin = user;
user = null; // user를 null로 덮어씁니다.

admin.sayThisName(); // John
admin.sayUserName(); // 에러
```

자유로운 this

- 자바스크립트에서 `this`는 런타임에 결정
- 객체가 없더라도 호출 가능
  - 엄격 모드에서 실행할 경우, `this == undefined`
  - 엄격 모드가 아니라면 `this`가 전역 객체를 참조
- 화살표 함수의 경우 자신만의 this를 가지지 않음
  - 객체 내부에서 선언하더라도 외부 컨텍스트에 있는 `this`를 가져오게 됨

```
let user = {
  firstName: "보라",
  sayHi() {
```

```
    let arrow = () => {
        alert(this.firstName); // sayHi의 this값을 가져옴
      }
    arrow();
  }
};

user.sayHi(); // 보라
```

- React에서는?
  - 메서드와 this를 비롯한 문법을 사용하지 않는 경우가 잦음
    - this 바인딩 문제
    - this를 사용할 필요가 없는 함수형 컴포넌트 권장
- this 명시적 바인딩은 추후 설명 예정

new 연산자와 생성자 함수

- `new`연산자와 생성자 함수로 유사한 객체 여러개를 만들 수 있음
- 생성자 함수
  - 대문자로 시작하도록 작명, `new`연산자를 붙여 실행

옵셔널 체이닝 '?.'

- `?.`는 '앞'의 평가 대상이 `undefined`나 `null`이면 평가를 멈추고 `undefined`를 반환
  - 프로퍼티가 없는 중첩 객체를 안전하게 접근 가능
  - 객체가 `null`이나 `undefined`여도 에러가 발생하지 않음

심볼형

- 객체 프로퍼티 키로 허용되는 유이한 자료형
- 심볼형을 사용하면, 외부 코드에서 접근이 불가능한 '숨김(hidden)' 프로퍼티 생성 가능
- 키가 심볼인 프로퍼티는 `for..in`반복문에서 배제됨

객체를 원시형으로 변환하기

- 원시형 자료는 문자형, 숫자형, boolean형 3가지로 변환 가능
- 객체는 논리평가 시 무조건 true이기에, 문자형 or 숫자형으로만 형 변환 가능
  - 문자형 : `alert(obj)`같이 객체를 출력하려고 할 때
  - 숫자형 : 객체끼리 빼는 연산을 하거나 수학 관련 함수를 적용할 때
    - `date1 - date2`같이 Date 객체끼리 빼는 연산을 하면, 두 날짜의 시간 차이가 반환됨

hint

- 객체 → 원시형 변환은 hint를 기준으로 세 종류로 구분 가능
  - `"string"` - `alert` 같이 문자열을 필요로 하는 연산
  - `"number"` - 수학 연산
  - `"default"` - 피연산자를 확신할 수 없는 경우

원시형 변환 알고리즘

1. 객체에 `obj[Symbol.toPrimitive](hint)`메서드가 있다면 호출
2. 해당 메서드가 없는데,
   a. hint가 `"string"`이라면 `obj.toString()`이나 `obj.valueOf()`를 호출
   b. hint가 `"number"`나 `"default"`라면 `obj.valueOf()`나 `obj.toString()`을 호출