24장 클로저

24.0 머릿말

- 클로저란
  - 클로저는 함수와 그 함수가 선언된 렉시컬 환경과의 조합이다.
  - 자바스크립트는 렉시컬 스코프(정적 스코프)를 따르는 프로그래밍 언어이다.
  - 
  - 예제

24.1 렉시컬 스코프

- 렉시컬 스코프(정적 스코프)란?
  - js엔진은 함수를 어디서 호출했는지가 아니라 어디에 정의했는지에 따라 상위 스코프를 결정한다.
    - 렉시컬 환경의 외부 렉시컬 환경에 대한 참조에 저장할 참조값, 즉 상위 스코프에 대한 참조는 함수 정의가 평가되는 시점에 함수가 정의된 환경(위치)에 의해 결정된다. 이것이 바로 렉시컬 스코프이다.
      - 
      - 예제
- 스코프의 실체는 실행 컨텍스트의 렉시컬 환경
  - 렉시컬 환경은 외부 렉시컬 환경에 대한 참조를 통해 상위 렉시컬 환경과 연결된다.
    - 이를 스코프 체인이라고 함

24.2 함수 객체의 내부 슬롯 [[Environment]]

- 함수는 자신의 내부 슬롯 [[Environment]]에 자신이 정의된 환경, 즉 상위 스코프의 참조를 저장한다.
  - 
  - 현재 실행중인 실행 컨텍스트의 렉시컬 환경을 저장한다.

24.3 클로저와 렉시컬 환경

- 
  - 예제
  - Outer 함수는 inner함수를 반환하고 생명 주기를 마감하여 실행 컨텍스트 스택에서 제거된다.
    - 이때 outer함수의 지역 변수 x 또한 생명주기가 마감된다
      - 하지만 여전히 내부함수가 지역변수 x를 참조하고 있다.
        - 이처럼 외부 함수보다 중첩 함수가 더 오래 유지되는 경우 중첩 함수는 이미 생명 주기가 종료된 외부 함수의 변수를 참조할 수 있다.
          - 이러한 중첩함수를 클로저라고 부른다.
    - 스택에서 제거된다고 해서 outer함수의 렉시컬 환경까지 소멸하는 것은 아니다.
      - 
      - 
      - 
      - 
      - outer함수의 렉시컬 환경은 inner함수의 [[Environment]] 내부 슬롯에 의해 참조되고 있고 inner함수는 전역 변수 innerFunc에 의해 참조되고 있으므로 가비지 컬렉션 대상이 되지 않기 때문
        - 가비지 컬렉터는 누군가가 참조하고 있는 메모리 공간을 함부로 해제하지 않는다.
- js의 모든 함수는 상위 스코프를 기억하므로 이론적으로 모든 함수는 클로저이다.
  - 하지만 모든 함수를 클로저라고 부르지 않는다.
    - 상위 스코프를 참조하고 외부 함수보다 생명주기가 긴 중첩함수를 클로저라고 부른다
      - 
      - 
      - 클로저가 아닌 예제
      - 
      - 클로저 예제
- 클로저에 의해 참조되는 상위 스코프의 변수를 자유 변수라고 한다.
  - 클로저란 자유 변수에 묶여있는 함수
- 상위 스코프의 식별자 중에서 기억해야 할 식별자만 기억하므로 불필요한 메모리 점유에 대해 걱정하지 않아도 된다.

24.4 클로저의 활용

- 클로저는 상태를 안전하게 변경하고 유지하기 위해 사용한다.
  - 상태를 안전하게 은닉하고 특정함수에게만 상태 변경을 허용하기 위해 사용
- 활용1
  - 
  - 안전하지 않은 코드 예제
  - 
  - num을 increase의 지역변수로 변경 하여 의도치 않은 상태 변경을 방지한 코드 예제 => 문제점: 이전 상태 유지 X
    - 
    - 클로저를 사용하여 이전 상태 유지하기 위한 예제
      - 이처럼 클로저는 상태가 의도치 않게 변경되지 않도록 안전하게 은닉하고 특정 함수에게만 상태 변경을 허용하여 상태를 안전하게 변경하고 유지하기 위해 사용한다.
- 활용2
  - 
  - 예제
  - 
  - 앞 예제를 생성자 함수로 표현한 코드 예제
- 활용3
  - 
  - 
  - 
  - 함수형 프로그래밍에서 클로저를 활용하는 코드 예제
    - 변수 값은 누군가에 의해 언제든지 변경될 수 있어 오류 발생의 근본적인 원인이 될 수 있다.
      - 외부 상태 변경이나 가변데이터를 피하고 불변성을 지향하는 함수형 프로그래밍에서 부수 효과를 최대한 억제하여 오류를 피하고 프로그램의 안정성을 높이기 위해 클로저는 적극적으로 사용된다.
    - 해당 예제에서는 increaser, decreaser 에 할당된 함수는 각각 자신만의 독립된 렉시컬 환경을 갖는다.
      - 그렇기에 counter 변수가 공유되지 않아 카운터 증감 연동이 되지 않는다.
        - 
        - 증감이 연동되게 하는 코드 예제

24.5 캡슐화와 정보 은닉

- 캡슐화란?
  - 캡슐화는 객체의 상태를 나타내는 프로퍼티와 프로퍼티를 참조하고 조작할 수 있는 동작인 메서드를 하나로 묶는 것을 말한다.
    - 캡슐화는 객체의 특정 프로퍼티나 메서드를 감출 목적으로 사용하기도 하는데 이를 정보 은닉이라 한다.
      - 정보은닉의 장점
        1. 접근을 제한하여 객체의 상태가 변경되는 것을 방지해 정보를 보호
        2. 객체 간의 상호 의존성, 즉 결합도를 낮추는 효과
- 대부분의 OOP 언어는 클래스를 정의하고 접근 제한자를 선언하여 공개 범위를 한정할 수 있다.
  - But, js는 public, private, protected 같은 접근 제한자를 제공하지 않는다.
    - 즉, 객체의 모든 프로퍼티와 메서드는 기본적으로 public하다.
      - 
      - 
      - 예제

24.6 자주 발생하는 실수

- 
  - 
  - 예제
  - let이나 const 키워드를 사용하는 반복문
    - for문, for...in문, for...of문, while문 등
    - 코드 블록을 반복 실행할 때마다 새로운 렉시컬 환경을 생성하여 반복할 당시의 상태를 마치 스냅숏을 찍는 것처럼 저장
      - 단, 이는 반복문의 코드 블록 내부에서 함수를 정의할 때 의미가 있다.
      - 반복문의 코드 블록 내부에 함수정의가 없는 경우 반복문이 생성하는 새로운 렉시컬 환경은 반복 직후 아무도 참조하지 않기때문에 가비지 컬렉션의 대상이 된다.