Access Modifiers & SourceMap

타입스크립트와 캡슐화, 소스맵

2024.06.22

두선아

Table of Contents

정보를 감추는 목적으로 private 사용하지 않기

클래스 멤버들을 정확하게 제한함으로써 예기치 않은 부작용을 줄이고, 의도하지 않은 접근을 방지할 수 있습니다.

타입스크립트의 접근 제어자

타입스크립트에서 접근 제어자(access modifier)는
클래스 멤버(속성 및 메서드)의 접근 가능성을 조절하는 데 사용됩니다.

```
class Example {
    public x: number;
    public greet() {
        console.log("Hello, world!");
    }
}
```

public

public 접근 제어자는
기본값입니다.

public 멤버는
클래스 외부에서도
접근할 수 있습니다.

```
class Example {
    private y: number;
    private greet() {
        console.log("Hello, world!");
    }
}
```

private

private으로 선언된 멤버는
외부에서 접근할 수 없습니다.

오직 클래스 내부에서만
접근할 수 있습니다.

```
class Parent {
    protected z: number;
    protected greet() {
        console.log("Hello, world!");
    }
}
```

protected

protected 멤버는
해당 클래스와 이를 상속하는
하위 클래스에서 접근할 수 있습니다.

인스턴스에서는 접근할 수 없습니다.

런타임 제약이 아닌, 컴파일 타임 제약

타입스크립트의 public, protected, private 접근 제어자를 사용하면
클래스 멤버의 공개 규칙을 강제할 수 있는 것으로 오해할 수 있습니다.

```
class Person {
  constructor(public name: string, protected age: number, private address: string) {}
}
```

- 컴파일 타임에서 발생하는 오류

```
let personExample = new Person('Alice', 30, '123 Main St');
console.log(personExample.name); // ⭕ public 멤버 접근 가능

// Property 'age' is protected and only accessible within class 'Person' and its subclasses
console.log(personExample.age); // ❌ protected 멤버는 클래스 외부에서 접근 불가

// Property 'address' is private and only accessible within class 'Person'.
console.log(personExample.address); // ❌ private 멤버는 클래스 외부에서 접근 불가
```

접근 제어자는?

타입스크립트 키워드이기 때문에 컴파일 후에는 제거됩니다.
ECMAScript 2022 이전에 사용하던 관례인 언더스코어 접두사나 마찬가지로 런타임에 효력이 없습니다.

- 컴파일된 자바스크립트 코드의 Person

```
class Person {
    constructor(name, age, address) {
        this.name = name;
        this.age = age;
        this.address = address;
    }
}
```

단언문을 사용하면?

런타임이 아닌 타입스크립트에서도 private 멤버에 접근할 수 있습니다.

```
(personExample as any).age;
(personExample as any).address;
```

캡슐화 예시A: Closure

```
declare function hash(text: string): number;

class PasswordChecker {
  checkPassword: (password: string) => boolean;

  constructor(passwordHash: number) {
    // checkPassword 함수가 클로저를 이용하여 passwordHash 변수에 접근할 수 있음
    this.checkPassword = (password: string) => {
      // 전달된 비밀번호를 해시화하여 저장된 해시값과 비교
      return hash(password) === passwordHash;
    };
  }
}

const checker = new PasswordChecker(hash("비밀번호"));
checker.checkPassword("비밀번호"); // 🙆
checker.checkPassword("다른 비밀번호"); // 🙅
```

hash 함수 호출
hash("비밀번호")가 123456이라는 해시 값을 반환한다고 가정합니다.

PasswordChecker 클래스의 인스턴스 생성
new PasswordChecker(123456) 호출로
PasswordChecker 클래스의 인스턴스가 생성됩니다.

생성자 함수가 호출되고 passwordHash 매개변수는
123456이라는 값을 받습니다.

클로저 생성
클로저는 함수가 생성될 당시의 lexical enviorment 을 기억합니다.

생성자 함수가 실행될 때, checkPassword 함수가 생성되고,
이 함수는 passwordHash를 참조합니다.

함수 실행
this.checkPassword 함수는 password 매개변수를 받아,
전달된 비밀번호를 해시화하고 저장된 passwordHash 값과 비교합니다.

const checker = new PasswordChecker(passwordHash);가 실행되고 나면
PasswordChecker 생성자 함수의 실행 컨텍스트는 스택에서 제거됩니다

checkPassword 함수는 생성될 때의 렉시컬 환경을 기억하여 클로저를 형성합니다.
이 렉시컬 환경에는 passwordHash가 있는 변수 객체가 포함됩니다.

즉, 클로저는 함수가 생성될 당시의 렉시컬 환경을 기억하여 나중에도 해당 환경에 접근할 수 있게 하며,
이렇게 참조되고 있는 변수는 가비지 컬렉팅되지 않습니다.

캡슐화 예시A의 단점

메서드 정의 위치가 제한적

생성자 외부에서 passwordHash에 접근할 수 없기 때문에
passwordHash에 접근하는 메서드도 생성자의 constructor 내부에 정의되어야 합니다.

메모리 낭비

인스턴스를 생성할 때마다 메서드의 복사본이 생성됩니다.

동일 클래스에서 생성된 인스턴스가 서로의 비공개 데이터에 접근하는 것이 불가능

클래스 단위 비공개란?
일반적인 객체지향 언어에서는 동일 클래스의 개별 인스턴스끼리 private 속성에 접근이 가능합니다.

캡슐화 예시B: Private Class Fields

ECMAScript 2022 표준, private 클래스 필드는 클래스 인스턴스의 외부에서는 직접 접근할 수 없습니다.

```
declare function hash(text: string): string;

class PasswordChecker {
  #passwordHash: string;

  constructor(passwordHash: string) {
    this.#passwordHash = passwordHash;
  }

  checkPassword(password: string) {
    return hash(password) === this.#passwordHash;
  }
}

const checker = new PasswordChecker(hash("비밀번호"));
checker.checkPassword("비밀번호"); // 🙆
checker.checkPassword("다른 비밀번호"); // 🙅
```

```
personExample.age;
personExample.address;
class PasswordChecker {
    constructor(passwordHash) {
        _PasswordChecker_passwordHash.set(this, void 0);
        __classPrivateFieldSet(this, _PasswordChecker_passwordHash, passwordHash, "f");
    }
    checkPassword(password) {
        return hash(password) === __classPrivateFieldGet(this, _PasswordChecker_passwordHash, "f");
    }
}
_PasswordChecker_passwordHash = new WeakMap();
```

이전 버전으로 자바스크립트 컴파일 시, this를 키로 하는 WeapMap으로 대체됩니다.

WeakMap은?

WeakMap은 키로 객체만 사용할 수 있습니다.

약한 참조 Weak References

WeakMap은 키로 사용된 객체에 대한 참조가 약하게 유지됩니다.
약한 참조는 다른 곳에서 해당 객체를 참조하지 않으면,
해당 객체가 가비지 컬렉션의 대상이 될 수 있음을 의미합니다.
즉, WeakMap의 키로 사용된 객체가 메모리에서 해제될 수 있습니다.

요약

public, protected, private와 같은
Typescript의 Access Modifiers는
타입 시스템에만 강제됩니다

런타임에는 소용이 없으며
단언문으로 우회할 수도 있습니다

자바스크립트에서 캡슐화를 구현하기 위해서는
클로저와 프라이빗 필드를 사용할 수 있습니다

소스맵을 사용하여 타입스크립트 디버깅하기

변환된 자바스크립트 코드

타입스크립트 코드를 실행한다는 것은?

타입스크립트 컴파일러가 생성한 자바스크립트 코드를 실행하는 것입니다.
기존 코드를 다른 형태로 변환하는 도구들도 마찬가지입니다. ex) 압축기 minifier, 전처리기 preprocessor

디버거는?

런타임에 동작합니다.
디버깅을 할 때 보게 되는 코드는 전처리기, 컴파일러, 압축기를 거친 복잡한 변환된 자바스크립트 코드입니다.

소스맵 source map

변환된 자바스크립트는 코드의 동작은 동일하지만, 형태가 매우 다르게 됩니다.
브라우저 제조사가 내놓은 해결책으로, 대부분의 브라우저와 많은 IDE가 지원합니다.
변환된 코드의 위치와 심벌들을 원본 코드의 원래 위치와 심벌로 매핑합니다.

```
"compilerOptions": {
  "sourceMap": true
}
```

tsconfig.json에서 sourceMap 설정

.ts 파일에 대응하는 .js와 .js.map 파일을 생성합니다.

소스맵이 js와 함께 있으면?

브라우저의 디버거에서 새로운 index.ts 파일이 나타납니다.
이제 ts 파일에 브레이크포인트 설정과 변수 조사가 가능합니다.

index.ts

디버거 좌측의 파일 목록에서 파일이 기울임 글꼴이다?
웹 페이지에 실제로 존재하는 파일이 아닙니다.

인라인으로 index.js.map 파일이 내용을 포함하게 하거나,
별도의 index.ts 파일을 가지도록 설정할 수 있습니다.
(참조 포함, 브라우저가 네트워크를 통해 로드)

✅ 체크 포인트

TS가 bundler나 minifier를 사용하고 있다면? 번들러나 압축기가 각자 소스맵을 생성합니다.

- 이상적인 디버깅 환경이 되려면 생성된 자바스크립트가 아닌, 원본 타입스크립트 소스로 매핑되도록 해야 합니다.

상용 환경에 소스맵이 유출되고 있는지 확인합니다.

- 디버거를 열지 않는 이상, 소스맵이 로드되지 않으므로 사용자에게 성능 저하는 없습니다.
- 원본 코드의 복사본에 주석, 버그 추적을 위한 url등 공개할 필요가 없는 내용이 있을 것이기 때문에 체크가 필요합니다.

NodeJS 디버깅에도 소스맵을 사용할 수 있습니다.

타입 체커가 코드를 실행하기 전 오류를 잡을 수 있지만, 디버거를 대체할 수는 없습니다.
소스맵을 사용해서 타입스크립트 디버깅 환경을 구축할 수 있습니다.

요약

원본 코드가 아닌 변환된 자바스크립트를 디버깅하지 말고
소스맵을 사용해서 런타임의 타입스크립트 코드를 디버깅하자~!

소스맵이 완전히 매핑되었는지 확인하자!

소스맵에 원본 코드가 그대로 포함되도록 설정되어 있을 수도 있습니다
공개하지 않도록 합시다

Thank you.

레퍼런스

이펙티브 타입스크립트 7장 - 코드를 작성하고 실행하기

타입스크립트 코드 실행: https://www.typescriptlang.org/play/