Ch16 프로퍼티 어트리뷰트

16.1 내부 슬롯과 내부 메서드

- ECMAScript 사양에 등잘하는 이중 대괄호([[...]])로 감싼 이름들이 내부 슬롯과 내부 메서드 이다. 내부 슬롯과 내부 메서드는 ECMAScript 사양에 정의된 대로 구현되어 자바스크립트 엔진에서 실제로 동작하지만 개발자가 직접 접근할 수 있도록 외부로 공개된 객체 프로퍼티는 아니다. 즉, 자바스크립트는 내부 슬롯과 내부 메서드에 직접적으로 접근할 수 있는 수단을 제공하지 않는다. 단, 일부 내부 슬롯과 내부 메서드에 한하여 간접적으로 접근할 수 있는 수단을 제공한다. 예를 들어 모든 객체는 [[Prototype]] 이라는 내부 슬롯을 갖는다. 내부 슬롯은 자바스크립트 엔진의 내부 로직이므로 원칙적으로 접근 할 수 없지만 [[Prototype]] 내부 슬롯의 경우, **proto** 를 통해 간접적으로 접근할 수 있다.

16.2 프로퍼티 어트리뷰트와 프로퍼티 디스크럽터 객체

- 자바스크립트 엔진은 프로퍼티를 생성할 때 프로퍼티의 상태를 나타내는 프로퍼티 어트리뷰트를 기본값으로 자동 정의한다.
- ES8에 도입된 Object.getOwnPropertyDescriptors 메서드는 모든 프로퍼티의 프로퍼티 어트리뷰트 정보를 제공하는 프로퍼티 디스크립터 객체들을 반환한다.

```
const person = {
  name: 'Lee'
};

// 프로퍼티 동적 생성
person.age = 20;

// 모든 프로퍼티의 프로퍼티 어트리뷰트 정보를 제공하는 프로퍼티 디스크립터 객체들을 반환한다.
console.log(Object.getOwnPropertyDescriptors(person));
/*
{
  name: {value: "Lee", writable: true, enumerable: true, configurable: true},
  age: {value: 20, writable: true, enumerable: true, configurable: true}
}
*/
```

16.3 데이터 프로퍼티와 접근자 프로퍼티

프로퍼티는 **데이터 프로퍼티**와 **접근자 프로퍼티**로 구분흰다.

데이터 프로퍼티

가지고 있는 프로퍼티 어트리뷰트

- value : 프로퍼티 키를 통해 프로퍼티 값에 접근하면 반환되는 값
- writable: 프로퍼티 값의 변경 가능 여부를 나타냄, false 일떄 읽기전용
- enumerable: 프로퍼티 열거 가능 여부를 나타냄, false 일때 for...in 문이나 Object.keys 메서드 등으로 열거할 수 없음
- configurable: 프로퍼티의 재정의 가능 여부를 나타냄, false 일 때 해당 프로퍼티의 삭제, 값 변경이 금지됨

초기엔 value 값 빼고 다 true로 초기화 된다.

```
const person = {
  name: 'Lee'
};

// 프로퍼티 동적 생성
person.age = 20;

console.log(Object.getOwnPropertyDescriptors(person));
/*
{
  name: {value: "Lee", writable: true, enumerable: true, configurable: true},
```

```
  age: {value: 20, writable: true, enumerable: true, configurable: true}
}
*/
```

접근자 프로퍼티

다른 데이터 프로퍼티의 값을 읽거나 저장할 때 사용한다.

- get : 접근자 프로퍼티를 통해 데이터 프로퍼티의 값을 읽을 때 호출되는 접근자 함수
- set : 접근자 프로퍼티를 통해 데이터 프로퍼티의 값을 저장할 때 호출되는 접근자 함수
- enumerable: 데이터 프로퍼티와 같음
- configurable: 데이터 프로퍼티와 같음

```
const person = {
  // 데이터 프로퍼티
  firstName: 'Ungmo',
  lastName: 'Lee',

  // fullName은 접근자 함수로 구성된 접근자 프로퍼티다.
  // getter 함수
  get fullName() {
    return `${this.firstName} ${this.lastName}`;
  },
  // setter 함수
  set fullName(name) {
    // 배열 디스트럭처링 할당: "31.1 배열 디스트럭처링 할당" 참고
    [this.firstName, this.lastName] = name.split(' ');
  }
};

// 데이터 프로퍼티를 통한 프로퍼티 값의 참조.
console.log(person.firstName + ' ' + person.lastName); // Ungmo Lee

// 접근자 프로퍼티를 통한 프로퍼티 값의 저장
// 접근자 프로퍼티 fullName에 값을 저장하면 setter 함수가 호출된다.
person.fullName = 'Heegun Lee';
console.log(person); // {firstName: "Heegun", lastName: "Lee"}

// 접근자 프로퍼티를 통한 프로퍼티 값의 참조
// 접근자 프로퍼티 fullName에 접근하면 getter 함수가 호출된다.
console.log(person.fullName); // Heegun Lee

// firstName은 데이터 프로퍼티다.
// 데이터 프로퍼티는 [[Value]], [[Writable]], [[Enumerable]], [[Configurable]] 프로퍼티 어트리뷰트를
갖는다.
let descriptor = Object.getOwnPropertyDescriptor(person, 'firstName');
console.log(descriptor);
// {value: "Heegun", writable: true, enumerable: true, configurable: true}

// fullName은 접근자 프로퍼티다.
// 접근자 프로퍼티는 [[Get]], [[Set]], [[Enumerable]], [[Configurable]] 프로퍼티 어트리뷰트를 갖는다.
descriptor = Object.getOwnPropertyDescriptor(person, 'fullName');
console.log(descriptor);
// {get: ƒ, set: ƒ, enumerable: true, configurable: true}
```

16.4 프로퍼티 정의

프로퍼티 정의란 새로운 프로퍼티를 추가하면서 프로퍼티 어트리뷰트를 명시적으로 정의하거나, 기존 프로퍼티의 프로퍼티 어트리뷰트를 재정의하는 것을 말한다.

Object.defineProperty 메서드를 사용하면 프로퍼티의 어트리뷰트를 정의할 수 있다.

```
const person = {};

// 데이터 프로퍼티 정의
Object.defineProperty(person, 'firstName', {
  value: 'Ungmo',
  writable: true,
  enumerable: true,
  configurable: true
});

Object.defineProperty(person, 'lastName', {
  value: 'Lee'
});

let descriptor = Object.getOwnPropertyDescriptor(person, 'firstName');
console.log('firstName', descriptor);
// firstName {value: "Ungmo", writable: true, enumerable: true, configurable: true}

// 디스크립터 객체의 프로퍼티를 누락시키면 undefined, false가 기본값이다.
descriptor = Object.getOwnPropertyDescriptor(person, 'lastName');
console.log('lastName', descriptor);
// lastName {value: "Lee", writable: false, enumerable: false, configurable: false}

// [[Enumerable]]의 값이 false인 경우
// 해당 프로퍼티는 for...in 문이나 Object.keys 등으로 열거할 수 없다.
// lastName 프로퍼티는 [[Enumerable]]의 값이 false이므로 열거되지 않는다.
console.log(Object.keys(person)); // ["firstName"]

// [[Writable]]의 값이 false인 경우 해당 프로퍼티의 [[Value]]의 값을 변경할 수 없다.
// lastName 프로퍼티는 [[Writable]]의 값이 false이므로 값을 변경할 수 없다.
// 이때 값을 변경하면 에러는 발생하지 않고 무시된다.
person.lastName = 'Kim';

// [[Configurable]]의 값이 false인 경우 해당 프로퍼티를 삭제할 수 없다.
// lastName 프로퍼티는 [[Configurable]]의 값이 false이므로 삭제할 수 없다.
// 이때 프로퍼티를 삭제하면 에러는 발생하지 않고 무시된다.
delete person.lastName;

// [[Configurable]]의 값이 false인 경우 해당 프로퍼티를 재정의할 수 없다.
// Object.defineProperty(person, 'lastName', { enumerable: true });
// Uncaught TypeError: Cannot redefine property: lastName

descriptor = Object.getOwnPropertyDescriptor(person, 'lastName');
console.log('lastName', descriptor);
// lastName {value: "Lee", writable: false, enumerable: false, configurable: false}

// 접근자 프로퍼티 정의
Object.defineProperty(person, 'fullName', {
  // getter 함수
  get() {
    return `${this.firstName} ${this.lastName}`;
  },
  // setter 함수
```

```
  set(name) {
    [this.firstName, this.lastName] = name.split(' ');
  },
  enumerable: true,
  configurable: true
});

descriptor = Object.getOwnPropertyDescriptor(person, 'fullName');
console.log('fullName', descriptor);
// fullName {get: ƒ, set: ƒ, enumerable: true, configurable: true}

person.fullName = 'Heegun Lee';
console.log(person); // {firstName: "Heegun", lastName: "Lee"}
```

프로퍼티 어트리뷰트 여러개를 한꺼번에 정의하고 싶다면?

Object.defineProperties 메서드를 이용하면 된다.

```
const person = {};

Object.defineProperties(person, {
  // 데이터 프로퍼티 정의
  firstName: {
    value: 'Ungmo',
    writable: true,
    enumerable: true,
    configurable: true
  },
  lastName: {
    value: 'Lee',
    writable: true,
    enumerable: true,
    configurable: true
  },
  // 접근자 프로퍼티 정의
  fullName: {
    // getter 함수
    get() {
      return `${this.firstName} ${this.lastName}`;
    },
    // setter 함수
    set(name) {
      [this.firstName, this.lastName] = name.split(' ');
    },
    enumerable: true,
    configurable: true
  }
});

person.fullName = 'Heegun Lee';
console.log(person); // {firstName: "Heegun", lastName: "Lee"}
```

16.5 객체 변경 방지

객체 확장 금지

- 메서드 : Object.preventExtensions
- 프로퍼티 추가 : X
- 프로퍼티 삭제: O

- 프로퍼티 값 읽기 : O
- 프로퍼티 값 쓰기 : O
- 프로퍼티 어트리뷰트 재정의 : O

요약: 프로퍼티 추가 금지, 근데 삭제는 가능

객체 밀봉

- 메서드 : Object.seal
- 프로퍼티 추가 : X
- 프로퍼티 삭제: X
- 프로퍼티 값 읽기 : O
- 프로퍼티 값 쓰기 : O
- 프로퍼티 어트리뷰트 재정의 : X

요약: 밀봉된 객체는 읽기와 쓰기만 가능

객체 동결

- 메서드 : Object.freeze
- 프로퍼티 추가 : X
- 프로퍼티 삭제: X
- 프로퍼티 값 읽기 : O
- 프로퍼티 값 쓰기 : X
- 프로퍼티 어트리뷰트 재정의 : X

요약: 동결된 객체는 읽기만 가능

근데 Obejct.freeze 메서드로 객체를 동결해도 중첩 객체까지 동결은 안된다.

객체의 중첩 객체까지 동결하여 변경이 불가능한 읽기 전용의 불변 객체를 구현하려면 객체를 값으로 갖는 모든 프로퍼티에 대해 재귀적으로 Object.freeze 메서드를 호출해야 한다.

```
function deepFreeze(target) {
  // 객체가 아니거나 동결된 객체는 무시하고 객체이고 동결되지 않은 객체만 동결한다.
  if (target && typeof target === 'object' && !Object.isFrozen(target)) {
    Object.freeze(target);
    /*
      모든 프로퍼티를 순회하며 재귀적으로 동결한다.
      Object.keys 메서드는 객체 자신의 열거 가능한 프로퍼티 키를 배열로 반환한다.
      ("19.15.2. Object.keys/values/entries 메서드" 참고)
      forEach 메서드는 배열을 순회하며 배열의 각 요소에 대하여 콜백 함수를 실행한다.
      ("27.9.2. Array.prototype.forEach" 참고)
    */
    Object.keys(target).forEach(key => deepFreeze(target[key]));
  }
  return target;
}

const person = {
  name: 'Lee',
  address: { city: 'Seoul' }
};

// 깊은 객체 동결
deepFreeze(person);

console.log(Object.isFrozen(person)); // true
// 중첩 객체까지 동결한다.
```

```
console.log(Object.isFrozen(person.address)); // true

person.address.city = 'Busan';
console.log(person); // {name: "Lee", address: {city: "Seoul"}}
```