**ECMA script 6**

**classes**

**function**

프로퍼티 getter와 setter

객체의 프로퍼티는 두 종류로 나뉩니다.

첫 번째 종류는 데이터 프로퍼티(data property) 입니다. 지금까지 사용한 모든 프로퍼티는 데이터 프로퍼티입니다. 데이터 프로퍼티 조작 방법에 대해선 모두 알고 계실 것이라 생각합니다.

두 번째는 접근자 프로퍼티(accessor property) 라 불리는 새로운 종류의 프로퍼티입니다. 접근자 프로퍼티의 본질은 함수인데, 이 함수는 값을 획득(get)하고 설정(set)하는 역할을 담당합니다. 그런데 외부 코드에서는 함수가 아닌 일반적인 프로퍼티처럼 보입니다.

## [getter와 setter](https://ko.javascript.info/property-accessors" \l "ref-401)

접근자 프로퍼티는 'getter(획득자)'와 ‘setter(설정자)’ 메서드로 표현됩니다. 객체 리터럴 안에서 getter와 setter 메서드는 get과 set으로 나타낼 수 있습니다.

let obj = {

get propName() {

// getter, obj.propName을 실행할 때 실행되는 코드

},

set propName(value) {

// setter, obj.propName = value를 실행할 때 실행되는 코드

}

};

getter 메서드는 obj.propName을 사용해 프로퍼티를 읽으려고 할 때 실행되고, setter 메서드는 obj.propName = value으로 프로퍼티에 값을 할당하려 할 때 실행됩니다.

프로퍼티 name과 surname이 있는 객체 user를 만들어봅시다.

let user = {

name: "John",

surname: "Smith"

};

이 객체에 fullName 이라는 프로퍼티를 추가해 fullName이 'John Smith'가 되도록 해봅시다. 기존 값을 복사-붙여넣기 하지 않고 fullName이 'John Smith'가 되도록 하려면 접근자 프로퍼티를 구현하면 됩니다.

let user = {

name: "John",

surname: "Smith",

get fullName() {

return `${this.name} ${this.surname}`;

}

};

alert(user.fullName); // John Smith

바깥 코드에선 접근자 프로퍼티를 일반 프로퍼티처럼 사용할 수 있습니다. 접근자 프로퍼티는 이런 아이디어에서 출발했습니다. 접근자 프로퍼티를 사용하면 함수처럼 호출 하지 않고, 일반 프로퍼티에서 값에 접근하는 것처럼 평범하게 user.fullName을 사용해 프로퍼티 값을 얻을 수 있습니다. 나머지 작업은 getter 메서드가 뒷단에서 처리해줍니다.

한편, 위 예시의 fullName은 getter 메서드만 가지고 있기 때문에 user.fullName=을 사용해 값을 할당하려고 하면 에러가 발생합니다.

let user = {

get fullName() {

return `...`;

}

};

user.fullName = "Test"; // Error (프로퍼티에 getter 메서드만 있어서 에러가 발생합니다.)

user.fullName에 setter 메서드를 추가해 에러가 발생하지 않도록 고쳐봅시다.

let user = {

name: "John",

surname: "Smith",

get fullName() {

return `${this.name} ${this.surname}`;

},

set fullName(value) {

[this.name, this.surname] = value.split(" ");

}

};

// 주어진 값을 사용해 set fullName이 실행됩니다.

user.fullName = "Alice Cooper";

alert(user.name); // Alice

alert(user.surname); // Cooper

이렇게 getter와 setter 메서드를 구현하면 객체엔 fullName이라는 '가상’의 프로퍼티가 생깁니다. 가상의 프로퍼티는 읽고 쓸 순 있지만 실제로는 존재하지 않습니다.

## [접근자 프로퍼티 설명자](https://ko.javascript.info/property-accessors" \l "ref-402)

데이터 프로퍼티의 설명자와 접근자 프로퍼티의 설명자는 다릅니다.

접근자 프로퍼티엔 설명자 value와 writable가 없는 대신에 get과 set이라는 함수가 있습니다.

따라서 접근자 프로퍼티는 다음과 같은 설명자를 갖습니다.

* **get** – 인수가 없는 함수로, 프로퍼티를 읽을 때 동작함
* **set** – 인수가 하나인 함수로, 프로퍼티에 값을 쓸 때 호출됨
* **enumerable** – 데이터 프로퍼티와 동일함
* **configurable** – 데이터 프로퍼티와 동일함

아래와 같이 defineProperty에 설명자 get과 set을 전달하면 fullName을 위한 접근자를 만들 수 있습니다.

let user = {

name: "John",

surname: "Smith"

};

Object.defineProperty(user, 'fullName', {

get() {

return `${this.name} ${this.surname}`;

},

set(value) {

[this.name, this.surname] = value.split(" ");

}

});

alert(user.fullName); // John Smith

for(let key in user) alert(key); // name, surname

프로퍼티는 접근자 프로퍼티(get/set 메서드를 가짐)나 데이터 프로퍼티(value를 가짐) 중 한 종류에만 속하고 둘 다에 속할 수 없다는 점을 항상 유의하시기 바랍니다.

한 프로퍼티에 get과 value를 동시에 설정하면 에러가 발생합니다.

// Error: Invalid property descriptor.

Object.defineProperty({}, 'prop', {

get() {

return 1

},

value: 2

});

## [getter와 setter 똑똑하게 활용하기](https://ko.javascript.info/property-accessors" \l "ref-403)

getter와 setter를 ‘실제’ 프로퍼티 값을 감싸는 래퍼(wrapper)처럼 사용하면, 프로퍼티 값을 원하는 대로 통제할 수 있습니다.

아래 예시에선 name을 위한 setter를 만들어 user의 이름이 너무 짧아지는 걸 방지하고 있습니다. 실제 값은 별도의 프로퍼티 \_name에 저장됩니다.

let user = {

get name() {

return this.\_name;

},

set name(value) {

if (value.length < 4) {

alert("입력하신 값이 너무 짧습니다. 네 글자 이상으로 구성된 이름을 입력하세요.");

return;

}

this.\_name = value;

}

};

user.name = "Pete";

alert(user.name); // Pete

user.name = ""; // 너무 짧은 이름을 할당하려 함

user의 이름은 \_name에 저장되고, 프로퍼티에 접근하는 것은 getter와 setter를 통해 이뤄집니다.

기술적으론 외부 코드에서 user.\_name을 사용해 이름에 바로 접근할 수 있습니다. 그러나 밑줄 "\_" 로 시작하는 프로퍼티는 객체 내부에서만 활용하고, 외부에서는 건드리지 않는 것이 관습입니다.

## [호환성을 위해 사용하기](https://ko.javascript.info/property-accessors" \l "ref-404)

접근자 프로퍼티는 언제 어느 때나 getter와 setter를 사용해 데이터 프로퍼티의 행동과 값을 원하는 대로 조정할 수 있게 해준다는 점에서 유용합니다.

데이터 프로퍼티 name과 age를 사용해서 사용자를 나타내는 객체를 구현한다고 가정해봅시다.

function User(name, age) {

this.name = name;

this.age = age;

}

let john = new User("John", 25);

alert( john.age ); // 25

그런데 곧 요구사항이 바뀌어서 age 대신에 birthday를 저장해야 한다고 해보겠습니다. age보다는 birthday가 더 정확하고 편리하기 때문이죠.

function User(name, birthday) {

this.name = name;

this.birthday = birthday;

}

let john = new User("John", new Date(1992, 6, 1));

이렇게 생성자 함수를 수정하면 기존 코드 중 프로퍼티 age를 사용하고 있는 코드도 수정해 줘야 합니다.

age가 사용되는 부분을 모두 찾아서 수정하는 것도 가능하지만, 시간이 오래 걸립니다. 게다가 여러 사람이 age를 사용하고 있다면 모두 찾아 수정하는 것 자체가 힘들죠. 게다가 age는 user 안에 있어도 나쁠 것이 없는 프로퍼티이기도 합니다.

기존 코드들은 그대로 두도록 합시다.

대신 age를 위한 getter를 추가해서 문제를 해결해 봅시다.

function User(name, birthday) {

this.name = name;

this.birthday = birthday;

// age는 현재 날짜와 생일을 기준으로 계산됩니다.

Object.defineProperty(this, "age", {

get() {

let todayYear = new Date().getFullYear();

return todayYear - this.birthday.getFullYear();

}

});

}

let john = new User("John", new Date(1992, 6, 1));

alert( john.birthday ); // birthday를 사용할 수 있습니다.

alert( john.age ); // age 역시 사용할 수 있습니다.