개체 프로토타입

프로토타입은 JavaScript 객체가 서로 기능을 상속하는 메커니즘입니다. 이 기사에서는 프로토타입이 무엇인지, 프로토타입 체인이 어떻게 작동하는지, 객체의 프로토타입을 설정하는 방법에 대해 설명합니다.

전제 조건:	JavaScript 함수 이해, JavaScript 기본 사항(첫 번째 단계 및 빌딩 블록 참조) 및 OOJS 기본 사항(개체 소개 참조)에 익숙합니다.
목적:	JavaScript 개체 프로토타입 이해, 프로토타입 체인 작동 방식 및 개체의 프로토타입 설정 방법.

프로토타입 체인

브라우저의 콘솔에서 객체 리터럴을 생성해 보십시오.

```
const myObject = {
  city: "Madrid",
  greet() {
    console.log(`Greetings from ${this.city}`);
  },
};

myObject.greet(); // Greetings from Madrid
```

city 이것은 하나의 데이터 속성 과 하나의 메서드 를 가진 객체입니다 greet() . 와 같이 개체 이름 *뒤에 마침표를* 콘솔에 입력하면 myObject. 콘솔이 이 개체에 사용할 수 있는 모든 속성 목록을 팝업으로 표시합니다. 뿐만 아니라 city 및 greet , 다른 많은 속성이 있음을 알 수 있습니다!

```
__defineGetter__
__defineSetter__
__lookupGetter__
__lookupSetter__
__proto__
city
constructor
greet
hasOwnProperty
isPrototypeOf
propertyIsEnumerable
toLocaleString
toString
valueOf
```

다음 중 하나에 액세스해 보세요.

```
myObject.toString(); // "[object Object]"
```

작동합니다(무엇을 하는지 명확하지 않더라도 toString()).

이러한 추가 속성은 무엇이며 어디에서 왔습니까?

JavaScript의 모든 객체에는 프로토타입 이라는 내장 속성이 있습니다 . 프로토타입은 그 자체로 객체이므로 프로토타입은 자체 프로토타입을 갖게 되어 프로토타입 체인 이라고 불리는 것을 만듭니다 . null 자체 프로토타입이 있는 프로토타입에 도달하면 체인이 종료됩니다 .

> 참고: 프로토타입을 가리키는 개체의 속성은 이라고 하지 않습니다 prototype . 그 이름은 표준이 아니지만 실제로는 모든 브라우저가 __proto__ . 개체의 프로토타입에 액세스하는 표준 방법은 메서드입니다 Object.getPrototypeOf() .

개체의 속성에 액세스하려고 할 때 개체 자체에서 속성을 찾을 수 없는 경우 프로토타입에서 속성을 검색합니다. 여전히 속성을 찾을 수 없으면 프로토타입의 프로토타입을 검색하고 속성을 찾거나 체인의 끝에 도달할 때까지 계속 검색합니다. 이 경우 undefined 반환됩니다.

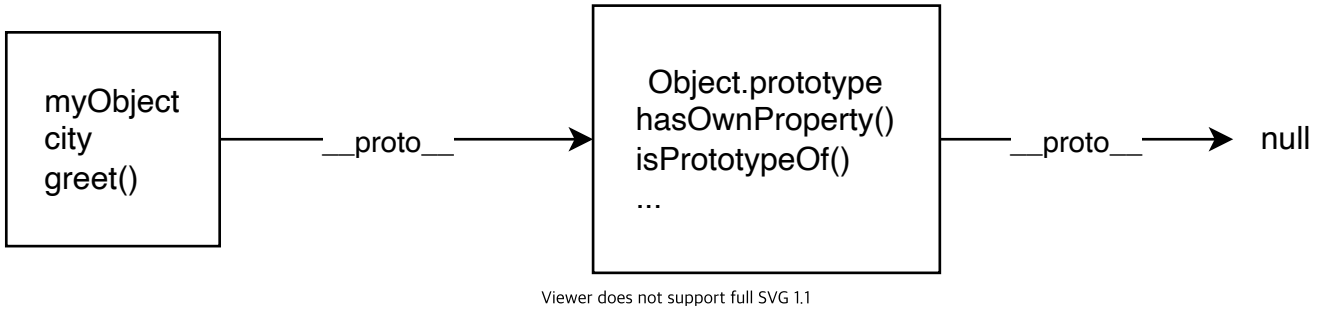
따라서 를 호출하면 myObject.toString() 브라우저는 다음을 수행합니다.

- toString 에서 찾는다 myObject
- 거기에서 찾을 수 없으므로 myObject for 의 프로토타입 개체에서 찾습니다. toString
- 거기에서 찾아서 호출합니다.

무엇을 위한 프로토타입입니까 myObject ? 알아보기 위해 다음 기능을 사용할 수 있습니다 Object.getPrototypeOf() .

```
Object.getPrototypeOf(myObject); // Object { }
```

이것이 이라는 객체 Object.prototype 이며, 모든 객체가 기본적으로 가지고 있는 가장 기본적인 프로토타입입니다. 의 프로토타입은 Object.prototype 이므로 null 프로토타입 체인의 끝에 있습니다.



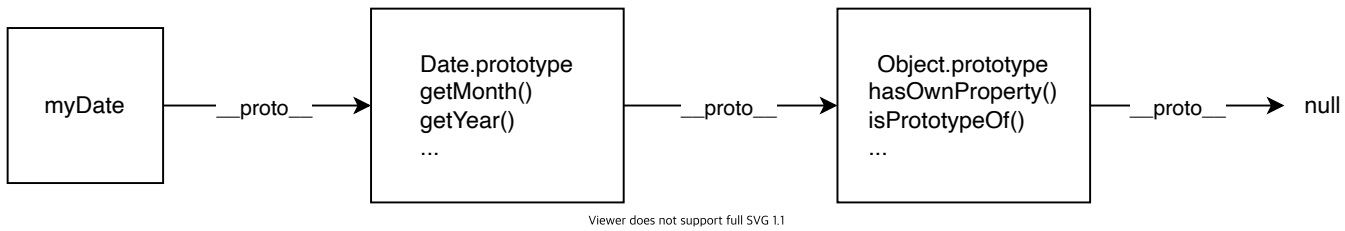
객체의 프로토타입이 항상 Object.prototype . 이 시도:

```
const myDate = new Date();
let object = myDate;

do {
  object = Object.getPrototypeOf(object);
  console.log(object);
} while (object);

// Date.prototype
// Object { }
// null
```

이 코드는 개체를 만든 Date 다음 프로토타입 체인을 따라 올라가 프로토타입을 기록합니다. myDate 의 프로토타입이 객체 이고 그 Date.prototype 프로토타입이 임을 보여줍니다 . Object.prototype



실제로 와 같은 친숙한 메서드를 호출하면 myDate2.getMonth() 에 정의된 메서드를 호출하는 것입니다 Date.prototype .

그림자 속성

동일한 이름의 속성이 개체의 프로토타입에 정의되어 있는 경우 개체에 속성을 정의하면 어떻게 됩니까? 보자:

```
const myDate = new Date(1995, 11, 17);

console.log(myDate.getYear()); // 95

myDate.getYear = function () {
  console.log("something else!");
};

myDate.getYear(); // 'something else!'
```

프로토타입 체인에 대한 설명이 주어지면 이는 예측 가능해야 합니다. 우리가 호출할 때 getYear() 브라우저는 먼저 myDate 해당 이름을 가진 속성을 찾고 프로토타입 myDate 이 정의되어 있지 않은 경우에만 프로토타입을 확인합니다. getYear() 따라서 에 추가하면 의 myDate 버전이 myDate 호출됩니다.

이를 속성 "섀도잉"이라고 합니다.

프로토타입 설정

JavaScript에서 객체의 프로토타입을 설정하는 다양한 방법이 있으며 여기서는 생성자 Object.create() 와 생성자 두 가지를 설명합니다.

Object.create 사용

이 Object.create() 메서드는 새 개체를 만들고 새 개체의 프로토타입으로 사용할 개체를 지정할 수 있습니다.

예를 들면 다음과 같습니다.

```
const personPrototype = {
  greet() {
    console.log("hello!");
  },
};

const carl = Object.create(personPrototype);
carl.greet(); // hello!
```

personPrototype 여기서는 메서드가 있는 개체를 만듭니다 greet() . 그런 다음 프로토타입 Object.create() 으로 새 개체를 만드는 데 사용합니다 . 이제 새 개체를 호출 personPrototype 할 수 있으며 프로토타입이 해당 구현을 제공합니다. greet()

생성자 사용

JavaScript에서 모든 함수에는 이라는 속성이 있습니다 prototype . 함수를 생성자로 호출하면 이 속성이 새로 생성된 개체의 프로토타입으로 설정됩니다(관례에 따라 라는 속성에서 __proto__).

prototype 따라서 생성자의 를 설정하면 해당 생성자로 생성된 모든 개체에 해당 프로토타입이 제공되도록 할 수 있습니다.

```
const personPrototype = {
  greet() {
    console.log(`hello, my name is ${this.name}!`);
  },
};

function Person(name) {
  this.name = name;
}

Object.assign(Person.prototype, personPrototype);
// or
// Person.prototype.greet = personPrototype.greet;
```

여기에서 다음을 생성합니다.

- 메소드 personPrototype 가 있는 객체 greet()
- Person() 생성할 사람의 이름을 초기화하는 생성자 함수 .

그런 다음 Object.assign을 사용하여 정의된 메서드를 함수의 속성 personPrototype 에 넣습니다 . Person prototype

이 코드 다음에 를 사용하여 생성된 개체는 자동으로 메서드를 포함하는 프로토타입으로 Person() 가져옵니다 . Person.prototype greet

```
const reuben = new Person("Reuben");
reuben.greet(); // hello, my name is Reuben!
```

이것은 또한 우리가 이전에 프로토타입이 myDate 호출된다고 말한 이유를 설명합니다 Date.prototype . 그것은 생성자 prototype 의 속성 입니다 Date .

소유 재산

위의 생성자를 사용하여 생성한 개체에는 Person 두 가지 속성이 있습니다.

- name 생성자에 설정되어 Person 개체 에 직접 표시되는 속성
- greet() 프로토타입에 설정된 메소드 .

메서드는 프로토타입에 정의되지만 데이터 속성은 생성자에 정의되는 이 패턴을 보는 것이 일반적입니다. 메서드는 일반적으로 우리가 생성하는 모든 개체에 대해 동일하지만 각 개체가 해당 데이터 속성에 대해 고유한 값을 갖기를 원하기 때문입니다(여기서 모든 사람이 다른 이름을 갖는 것처럼).

여기서 와 같이 객체에 직접 정의된 속성을 소유 속성 name 이라고 하며 정적 메서드를 사용하여 속성이 자체 속성인지 확인할 수 있습니다 . Object.hasOwn()

```
const irma = new Person("Irma");

console.log(Object.hasOwn(irma, "name")); // true
console.log(Object.hasOwn(irma, "greet")); // false
```

> 참고: 여기서 비정적 방법을 사용할 수도 있지만 가능하면 Object.hasOwnProperty() 사용하는 것이 좋습니다 . Object.hasOwn()

프로토타입 및 상속

프로토타입은 JavaScript의 강력하고 매우 유연한 기능으로, 코드를 재사용하고 객체를 결합할 수 있습니다.

특히 상속 버전을 지원합니다 . 상속은 프로그래머가 시스템의 일부 객체가 다른 객체의 보다 전문화된 버전이라는 생각을 표현할 수 있게 해주는 객체 지향 프로그래밍 언어의 기능입니다.

예를 들어 학교를 모델링하는 경우 *교수* 와 *학생이* 있을 수 있습니다 . 둘 다 *사람* 이므로 몇 가지 공통된 기능(예: 둘 다 이름이 있음)이 있지만 각각 추가 기능을 추가할 수 있습니다(예: 교수 가르치는 과목이 있거나) 동일한 기능을 다른 방식으로 구현할 수 있습니다. OOP 시스템에서 우리는 교수와 학생이 모두 사람 에게서 물려받았다고 말할 수 있습니다 .

Professor JavaScript에서 개체 가 프로토타입을 Student 가질 수 있는 경우 Person 공통 속성을 상속하는 동시에 달라야 하는 속성을 추가하고 재정의하는 방법을 볼 수 있습니다 .

다음 기사에서는 객체 지향 프로그래밍 언어의 다른 주요 기능과 함께 상속에 대해 논의하고 JavaScript가 이를 지원하는 방법을 살펴봅니다.

요약

이 기사에서는 프로토타입 객체 체인을 통해 객체가 서로 기능을 상속하는 방법, 프로토타입 속성, 생성자에 메서드를 추가하는 데 사용할 수 있는 방법 및 기타 관련 항목을 포함하여 JavaScript 객체 프로토타입을 다뤘습니다.