JS Fundamentals
Prototype

Автор презентации: Макухина Марина

Объектно-ориентированное программирование

ООП — это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Фундаментальная идея ООП призвана отразить наше понимание естественного мира. Например, можно рассмотреть любой транспорт как объект: с набором свойств и методов. Кроме того, ООП позволяет думать о вещах абстрактно (*транспорт*) и конкретно (о*пределенное транспортное средство*).

Классы в ООП

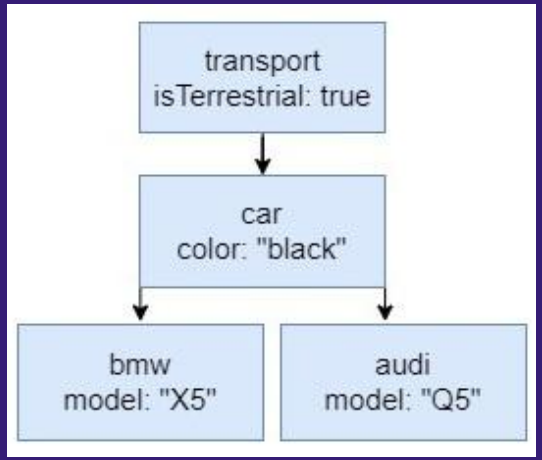
Класс — это шаблон для создания объектов, который содержит определенный набор свойств и методов. Представители классов в JavaScript — это функции-конструкторы.

```
function Transport(isTerrestrial) {
  this.isTerrestrial = isTerrestrial;
}
let plane = new Transport(false);
alert(plane instanceof Transport); // true
```

Класс Transport может содержать характеристику *тип передвижения*. Созданный на основе этого класса объект будет содержать конкретную информацию. Оператор *instanceof* позволяет проверить принадлежность объекта к определенному классу.

Прототипы

Прототипы — это механизм, с помощью которого объекты JavaScript наследуют свойства друг от друга. Объекты имеют специальное скрытое свойство обозначенное в спецификации *[[Prototype]]* , которое либо равно *null*, либо ссылается на другой объект. При попытке прочитать некоторое свойство, если оно отсутствует у объекта, JavaScript попробует найти его в цепочке прототипов.



То есть для объектов *audi* и *bmw* при обращение к свойству *isTerrestrial* вернется значение *true*. Такой механизм называется «прототипное наследование».

Способы создания прототипов

Свойство *__proto__* — это геттер/сеттер для *[[Prototype]]*.

```
let car = { color: "black" };
let bmw = { model: "X5" };
bmw.__proto__ = car;
console.log(bmw.color);  // black
```

Метод *create()* создает новый объект из указанного объекта-прототипа.

```
let audi = Object.create(car);
audi.["model"] = "Q5";

console.log(audi.color);  // black
```

Создания прототипа через свойство F.prototype

Свойство *F.prototype* устанавливает *[[Prototype]]* для новых объектов при вызове *new F()*. Свойство "*prototype*" является особым, только когда оно назначено функции-конструктору, которая вызывается оператором *new*. В других объектах prototype будет обычным свойством.

```
function Toyota(modelName) {
  this.model = modelName;
}
let car = { color: "white" }
Toyota.prototype = car;
let toyota = new Toyota("Camry");
console.log(toyota.color);  // white
```

Обратите внимание, что прототип был установлен перед определением нового объекта. После создания *F.prototype* может измениться, и новые объекты, в отличии от старых, будут иметь другой объект в качестве *[[Prototype]]*.

Классы

До стандарта ES6 создание классов в JavaScript было возможно только с помощью функций-конструкторов. Теперь появился новый удобный синтаксис создания классов.

Когда вызывается *new Toyota ("Camry")*:

- создается новый объект;
- constructor запускается с заданным аргументом и сохраняет его в *this.model*;
- методы сохраняются в *Toyota.Prototype*.

```
class Toyota {
  constructor(modelName) {
    this.model = modelName;
  }
  modelInUpper() {
    return this.model.toUpperCase();
  }
}

let toyota = new Toyota("Camry");
console.log(toyota.modelInUpper());
// CAMRY
```

Разница функций-конструкторов и классов

Этот код подобен реализации с помощью класса, но есть отличия:

- методы класса являются неперечисляемыми. Определение класса устанавливает флаг *enumerable* в *false* для всех методов в *prototype*;
- код внутри класса автоматически находится в строгом режиме.

```
function Toyota(modelName) {
  this.model = modelName;

  Toyota.prototype.modelInUpper = function ()
{
    return this.model.toUpperCase();
  };
}

let toyota = new Toyota("Land Cruiser Prado");
console.log(toyota.modelInUpper());
// LAND CRUISER PRADO
```

Встроенные прототипы

Встроенные прототипы имеют объекты *Array*, *Object*, *Boolean*, *Date*, *Function*, *RegExp*, *String* и *Number*. Наверху иерархии встроенных прототипов находится Object.prototype.

```
let obj = {};
let arr = [1, 2];
console.log(obj.toString());
// [object Object]
console.log(arr.toString());  // 1,2
```

Когда создается объект, свойство *[[Prototype]]* этого объекта устанавливается на *Object.prototype*. Если был создан массив, его прототипом устанавливается *Array.prototype*.

У *Object.prototype* есть свой метод *toString()*, но так как *Array.prototype* ближе в цепочке прототипов, то берётся именно вариант для массивов.

Изменение встроенных прототипов

Встроенные прототипы можно изменять, но это является плохой практикой. Это бывает необходимо в случае создание полифилов. Полифил — это код, реализующий какую-либо функциональность, которая существует в спецификации JavaScript, но ещё не поддерживается текущим движком JavaScript.

```
if (!String.prototype.includes) {
 String.prototype.includes = function
(search, start) {
  "use strict";
  if (typeof start !== "number") {
    start = 0;
  }
  return start + search.length > this.length
    ? false
    : this.indexOf(search, start) !== -1;
 };
}
```

Объекты-обертки для примитивных типов

При попытке получить доступ к свойству примитивного типа, будет создан временный объект *String*, *Number* или *Boolean*. Таким образом, будет вызван метод из прототипа *String.prototype*, *Number.prototype* или *Boolean.prototype*. После чего, сборщик мусора удаляет временный объект.

```
String.prototype.returnMe = function ()
{
  return this;
};

let primitive = "text";
let obj = primitive.returnMe();
console.log(typeof primitive); // string
console.log(typeof obj); // object
```

Значения *null* и *undefined* не имеют объектов-обёрток.

Особенности цикла for..in и оператора in

Если перебирать содержимое объекта в цикле *for..in*, в перечисление будут включены все свойства, достижимые по цепочке.

```
let transport = { isTerrestrial: true };
let car = { color: "black" };
car.__proto__ = transport;
for (let value in car) {
 console.log(value); // color isTerrestrial
}
```

```
let pr =
Object.getOwnPropertyNames(car);
for (let value of pr) {
 console.log(value);
 // color
}
```

Оператор *in*, используемый для проверки существования свойства объекта, проверит всю цепочку объекта.

```
console.log("isTerrestrial" in car); // true
```

Разница классического и прототипного наследования

Классическое наследование

Экземпляры наследуются от абстракции (класса), и создают отношения между подклассами.

Создает иерархию классов, как побочный, которые ведут к застывшему коду (трудно изменить) и хрупкости (легко сломать из-за возникновения побочных эффектов при изменении базовых классов).

Прототипное наследование

Экземпляры наследуются от других экземпляров.

Экземпляры могут состоять из множества различных исходных объектов, что обеспечивает легкое выборочное наследование.

Полезные ссылки

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes

https://learn.javascript.ru/native-prototypes

https://learn.javascript.ru/class

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Enumerability_and_ownership_of_properties

На этом всё!

•••

Спасибо за внимание