**НАСЛЕДОВАНИЕ**

Основные понятия

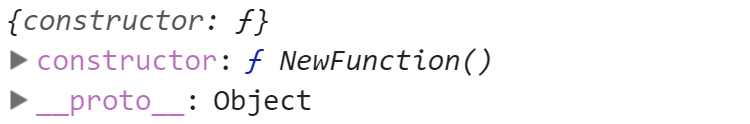
Из предыдущей сессии необходимо помнить три ключевых момента.

1. При вызове метода у объекта происходит его поиск с начала у самого объекта, затем у прототипа этого объекта.
2. Прототип уже созданного объекта доступен по ссылке \_\_proto\_\_. И вы можете при необходимости его задать, установив значение в это поле.
3. Если вы планируете создавать объект с помощью функции конструктора, то вы устанавливаете необходимый вам прототип в свойство prototype функции-конструктора. Затем, при создании объекта значение в \_\_proto\_\_ подставляется из свойства prototype функции-конструктора.

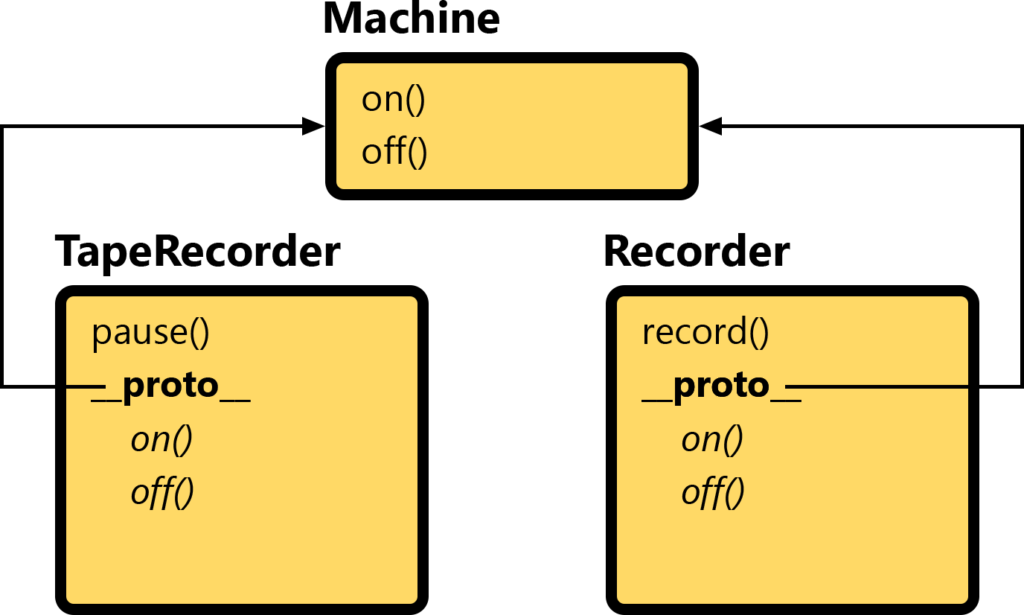
Записав новую функцию-конструктор и выведя ее свойство prototype в консоль, можно обнаружить, что по умолчанию, там находится не только ссылка на прототип объекта Object, а и свойство constructor.

function NewFunction() {}

console.log(NewFunction.prototype);

  
Свойство constructor – ссылка на функцию, создавшую экземпляр объекта. То есть в данном случае, по сути на саму себя.

Прототипное наследование в JavaScript

Наследование – это концепция ООП, благодаря которой вы можете расширить функциональность одного класса за счет методов и свойств другого класса. Другими словами, некий класс «наследник» имеет возможность вызывать и использовать методы «родителя». При этом, фактически у самого «наследника» этих методов нет. С помощью специального механизма наследования он их вызывает у «родителя» как будто они принадлежат ему самому.  
Класс, определённый через наследование от другого класса, называется: производным классом, классом потомком (derived class) или подклассом (subclass). Класс, от которого новый класс наследуется, называется: предком (parent), базовым классом (base class) или суперклассом(parent class).  
Напомним, что классов в JavaScript нет, условные «классы» реализуются с помощью функций-конструкторов.  
Механизм наследования дает возможность получить доступ к функциональности родительских объектов («классов»), благодаря чему, можно легко повторно использовать код и расширять функциональность.  
Схематически наследование можно отобразить так.  


Экземпляры объекта могут как использовать унаследованную функциональность, так и иметь свою собственную.  
Создадим родительский объект Machine, который будет унаследован и дочерний объект TapeRecorder, который будет наследовать.

function Machine(product) {

this.product = product;

}

Machine.prototype.on = function() {

console.log(`${this.product} is ON!`);

}

Machine.prototype.off = function() {

console.log(`${this.product} is OFF!`);

}

function TapeRecorder(product) {

this.product = product;

}

Реализация наследования осуществляется с помощью одной строки кода.

TapeRecorder.prototype = Object.create(Machine.prototype);

Метод Object.create() создаёт новый объект и устанавливает в его прототип то, что было указано в качестве аргумента.  
Теперь можно использовать методы родительского объекта.

function Machine(product) {

this.product = product;

}

Machine.prototype = {

on: function() {

console.log(`${this.product} is ON!`);

},

off: function() {

console.log(`${this.product} is OFF!`);

}

}

function TapeRecorder(product) {

this.product = product;

}

TapeRecorder.prototype = Object.create(Machine.prototype);

const tapeRecorder = new TapeRecorder('Tape Recorder');

tapeRecorder.on(); // Tape Recorder is ON!

Обратите внимание, в данном примере был использован иной синтаксис помещения методов в прототип. Оба способа определения методов в прототип допустимы, разница только в синтаксисе.  
После того, как был установлен прототип для дочернего объекта TapeRecorder с помощью Object.create (инициализирован механизм наследования), в его прототип можно также добавить дополнительные методы. Но запомните, делать это стоит только после первоначальной установки прототипа (Object.create), в ином случае, методы, которые были определенны в прототип дочернего объекта до инициализации наследования, будут затёрты.

function TapeRecorder(product) {

this.product = product;

}

TapeRecorder.prototype = Object.create(Machine.prototype);

TapeRecorder.prototype.pause = function() {

console.log(`${this.product} on PAUSE!`);

}

const tapeRecorder = new TapeRecorder('Tape Recorder');

tapeRecorder.on(); // Tape Recorder is ON!

tapeRecorder.pause(); // Tape Recorder on PAUSE!

Свойство constructor и переопределение методов

Так как в свойство prototype, дочернего объекта, устанавливается как прототип родительский объект, то свойство constructor (которое автоматически устанавливается и ссылается на функцию-конструктор, создавшую экземпляр) будет перезаписано родительским свойством constructor.

TapeRecorder.prototype.constructor === Machine // true

TapeRecorder.prototype.constructor === TapeRecorder // false

Для сохранения корректного конструктора, необходимо его явно присвоить вручную.

TapeRecorder.prototype.constructor = TapeRecorder;

TapeRecorder.prototype.constructor === Machine // false

TapeRecorder.prototype.constructor === TapeRecorder // true

На данном этапе, реализация наследования выглядит следующим образом.

function Machine(product) {

this.product = product;

}

Machine.prototype.on = function() {

console.log(`${this.product} is ON!`);

}

Machine.prototype.off = function() {

console.log(`${this.product} is OFF!`);

}

function TapeRecorder(product) {

this.product = product;

}

TapeRecorder.prototype = Object.create(Machine.prototype);

TapeRecorder.prototype.constructor = TapeRecorder;

TapeRecorder.prototype.pause = function() {

console.log(`${this.product} on PAUSE!`);

}

const tapeRecorder = new TapeRecorder('Tape Recorder');

tapeRecorder.on(); // Tape Recorder is ON!

tapeRecorder.pause(); // Tape Recorder on PAUSE!

Вернемся к понятию цепочка прототипов. Напомним, при вызове метода, его поиск сначала происходит непосредственно среди методов, принадлежащих самому объекту. Если необходимый метод не был найден, то его поиск продолжается в прототипе объекта. И так до самого верхнего уровня, которым является глобальный объект Object.  
Родительские методы, можно переопределять в дочерних объектах.

function Machine(product) {

this.product = product;

}

Machine.prototype.on = function() {

console.log(`${this.product} is ON!`);

}

function TapeRecorder(product, model) {

this.product = product;

}

TapeRecorder.prototype = Object.create(Machine.prototype);

TapeRecorder.prototype.constructor = TapeRecorder;

TapeRecorder.prototype.on = function() {

console.log(`${this.product} is ON! Music is playing!`);

}

const tapeRecorder = new TapeRecorder('Tape Recorder');

tapeRecorder.on(); // Tape Recorder is ON! Music is playing!

Как видно из примера, родительский метод был переопределен и, так как необходимый метод был найден в дочернем объекте, он сразу был вызван из него и поиск далее прекратился.  
Полная реализация наследования выглядит так:

function Machine(product) {

this.product = product;

}

Machine.prototype.on = function() {

console.log(`${this.product} is ON!`);

}

Machine.prototype.off = function() {

console.log(`${this.product} is OFF!`);

}

function TapeRecorder(product, model) {

this.product = product;

}

TapeRecorder.prototype = Object.create(Machine.prototype);

TapeRecorder.prototype.constructor = TapeRecorder;

TapeRecorder.prototype.on = function() {

console.log(`${this.product} is ON! Music is playing!`);

}

TapeRecorder.prototype.pause = function() {

console.log(`${this.product} on PAUSE!`);

}

const tapeRecorder = new TapeRecorder('Tape Recorder');

tapeRecorder.on(); // Tape Recorder is ON! Music is playing!

tapeRecorder.pause(); // Tape Recorder on PAUSE!

Родительские свойства и методы в контексте дочернего объекта

Если есть необходимость расширить дочерний конструктор свойствами родительского, можно вызвать родительский конструктор в контексте дочернего. Таким образом, все свойства родительского объекта будет присвоены в this дочернего и без проблем отработают.

function TapeRecorder(product, model) {

Machine.apply(this, arguments);

this.model = model;

}

TapeRecorder.prototype = Object.create(Machine.prototype);

TapeRecorder.prototype.constructor = TapeRecorder;

TapeRecorder.prototype.pause = function() {

console.log(`${this.product} on PAUSE!`);

}

const tapeRecorder = new TapeRecorder('Tape Recorder', 'TH-45');

tapeRecorder.on(); // Tape Recorder is ON!

Как видно из примера выше, явно не указав this.product, это свойство было получено из родительского объекта.  
Бывают случаи, когда необходимо не просто переопределить родительский метод, а расширить его функциональность.  
Это можно реализовать с помощью вызова родительского метода напрямую из прототипа.

function Machine(product) {

this.product = product;

}

Machine.prototype.on = function() {

console.log(`${this.product} is ON!`);

}

function TapeRecorder(product, model) {

Machine.apply(this, arguments);

this.model = model;

}

TapeRecorder.prototype = Object.create(Machine.prototype);

TapeRecorder.prototype.constructor = TapeRecorder;

TapeRecorder.prototype.on = function() {

Machine.prototype.on.apply(this, arguments);

console.log(`Model: ${this.model}.`);

}

const tapeRecorder = new TapeRecorder('Tape Recorder', 'TH-45');

tapeRecorder.on();

// Tape Recorder is ON!

// Model: TH-45.

Посмотрим на финальную версию наследования, в которой реализованы все основные подходы для его правильного функционирования.

function Machine(product) {

this.product = product;

}

Machine.prototype.on = function() {

console.log(`${this.product} is ON!`);

}

Machine.prototype.off = function() {

console.log(`${this.product} is OFF!`);

}

function TapeRecorder(product, model) {

Machine.apply(this, arguments);

this.model = model;

}

TapeRecorder.prototype = Object.create(Machine.prototype);

TapeRecorder.prototype.constructor = TapeRecorder;

TapeRecorder.prototype.on = function() {

Machine.prototype.on.apply(this, arguments);

console.log(`Model: ${this.model}.`);

}

TapeRecorder.prototype.pause = function() {

console.log(`${this.product} on PAUSE!`);

}

const tapeRecorder = new TapeRecorder('Tape Recorder', 'TH-45');

tapeRecorder.on();

// Tape Recorder is ON!

// Model: TH-45.

Кроме всего прочего, для удобства реализации наследования, можно использовать специальную функцию. Без использования такой функции необходимо сначала описать родительскую функцию конструктор, далее дочернюю функцию конструктор, но без методов в прототипе, далее реализовать наследование и только потом добавлять в дочернюю функцию дополнительные методы.

function inheritance(parent, child) {

let tempChild = child.prototype;

child.prototype = Object.create(parent.prototype);

child.prototype.constructor = child;

for (let key in tempChild) {

if (tempChild.hasOwnProperty(key)) {

child.prototype[key] = tempChild[key];

}

}

}

Функция делает следующее.

* • все содержимое, которое было в прототипе наследника, переносится во временный объект;
* • затем реализуем наследование так, как мы это делали раньше, однако эта операция полностью затрёт все существовавшие в дочернем прототипе методы;
* • благодаря тому, что копия изначального прототипа дочерней функции конструктора была сохранена во временную переменную, возвращаем все методы из временной копии с помощью цикла.

Метод hasOwnProperty проверяет, содержит ли объект указанное свойство в качестве собственного свойства объекта.  
Благодаря этой функции вы можете описать 2 класса полностью независимо друг от друга.

function inheritance(parent, child) {

let tempChild = child.prototype;

child.prototype = Object.create(parent.prototype);

child.prototype.constructor = child;

for (let key in tempChild) {

if (tempChild.hasOwnProperty(key)) {

child.prototype[key] = tempChild[key];

}

}

}

function Machine(product) {

this.product = product;

}

Machine.prototype = {

on: function() {

console.log(`${this.product} is ON!`);

},

off: function() {

console.log(`${this.product} is OFF!`);

}

}

function TapeRecorder(product, model) {

this.product = product;

this.model = model;

}

TapeRecorder.prototype = {

on: function() {

console.log(`Model: ${this.model}.`);

},

pause: function() {

console.log(`${this.product} on PAUSE!`);

}

}

inheritance(Machine, TapeRecorder);

const tapeRecorder = new TapeRecorder('Tape Recorder', 'TH-45');

tapeRecorder.on(); // Model: TH-45.

[ПРЕДЫДУЩАЯ ЧАСТЬ](https://www.jsexpert.net/course/clear-javascript-advanced/?course_type=content&course_page=2&lecture=1)[СЛЕДУЮЩАЯ ЧАСТЬ](https://www.jsexpert.net/course/clear-javascript-advanced/?course_type=content&course_page=2&lecture=3&section-quiz=1)