Конструкторы

Литерал позволяет создать один объект. Для того, чтоб создать множество однотипных объектов с различиями в значениями можно использовать клонирование и изменять, то что нужно, но это слишком медленно и затратно.

Правильным решением будет использование *конструкторов* – это функция, которая возвращает новый объект. Рассмотрим простой пример:

function User(name) {

this.name = name;

this.isAdmin = false;

}

vasy = new User('Вася');

console.log (vasy);

Функция конструктор должна называться с заглавной буквы, а для создания нового объекта используется ключевое слово new.

Эта функция создала новый объект, который никак не отличается от объектов созданных литералом. На самом деле любая функция может конструктором, если ее вызвали с new.

Если вы разрабатываете библиотеку, то можно создать функцию, которая создаст объект и без new:

function User(name) {

if(!new.target) {

return new User(name);

}

this.name = name;

this.isAdmin = false;

}

vasy = User('Вася');

console.log(vasy);

Здесь используется свойство каждой функции - new.target, он появляется, если вызван с new.

Как и в любой функции можно вернуть значение, притом если вернуть объект, то он полностью перекроет:

function Animal() {

this.type = 'Бонобо'

return {

type: 'Человек'

}

}

console.log (new Animal ());

Конечно, в конструкторах можно указать методы:

function User(name) {

this.name = name;

this.isAdmin = false;

this.like = function () {

alert('Like It!');

}

}

vasy = new User('Вася');

vasy.like ();

Прототипирование объекта

У нас есть объект animal, которая содержит два метода. Есть объект bear, который «расширяет» animal одним своим свойством, притом bear – это отдельная сущность. Точно такой же объект rabbit.

Для того чтоб расширить один объект другим объектом используется *прототипное наследование* – это когда объект расширяется другим объектом путем полного копирования в *прототип* объекта его свойств. Часть с расширением можно перефразировать, как объект наследует свойства другого объекта.

[[Prototype]]– это скрытое свойство, в которое вкладываются все свойства объекта родителя. Для того чтоб установить прототип в объект есть свойство \_\_proto\_\_, в который надо указать объект-родитель.

let animal = {

walk() {

console.log ('...walk');

},

eat() {

console.log ('eat...');

}

}

let bear = {

name: 'Bear',

roar() {

console.log ('ROAR');

},

\_\_proto\_\_: animal

}

let rabbit = {

jump() {

console.log ('jump')

},

\_\_proto\_\_: animal

}

bear.roar ()

bear.eat ();

bear.walk ();

Если воспользоваться console.dir, то можно увидеть свойство \_\_proto\_\_, а в нем и все свойство из animal. Но в самом прототипе лежит еще один прототип. Вообще у каждого объект есть прототип и притом это Object.

Можно расширять, но можно перекрывать, изменим bear:

let bear = {

name: 'Bear',

roar() {

console.log('ROAR');

},

toString() {

return 'hungry bear';

},

\_\_proto\_\_: animal

}

Теперь вместо стандартного выведется hungry bear. Притом toString просто добавился в bear, а в самом прототипе лежит стандартная реализация toString.

Произошло *перекрытие* объявленного toString в объекте bear. Просто, когда вызывается свойство, то сначала оно будет искаться в самом объекте, а лишь после в его прототипе.

Вообще объекты и прототипы следует рассматривать, как две разные сущности, которые могут просто доставать свойство друг - друга, поэтому свойств у прототипа есть особенности:

* Их нельзя удалить и изменить
* Прототипы не имеют собственного контекста this
* Если прототип использует перекрытые свойства, то изначальные игнорируются.

Пример:

let animal = {

walk() {

console.log ('...walk');

},

eat() {

console.log ('eat...');

},

toString() {

return `${this.name}`;

}

}

let bear = {

name: 'Bear',

roar() {

console.log ('ROAR');

},

\_\_proto\_\_: animal

}

let rabbit = {

name: 'Rabbit',

jump() {

console.log ('jump')

},

\_\_proto\_\_: animal

}

bear.toString ();

rabbit.toString ();

Это все лежит в основе прототипирования, но чаще всего используются конструкторы, а там немного другой подход.

Прототипирование конструктора

У конструкторов есть возможность указать прототип для нового объекта, через свойство у функции prototype.

let animal = {

walk() {

console.log ('...walk');

},

eat() {

console.log ('eat...');

},

toString() {

return `${this.name}`;

}

}

function Rabbit() {

this.name = 'Rabbit',

this.jump = function () {

console.log ('jump');

}

}

Rabbit.prototype = animal;

let r = new Rabbit();