Objects: creation, literal, “new” operator, prototyping, \_\_proto\_\_, inheritance, constructor, levels of visibility

Creation

Способы создания объекта:

* Оператор new
* Литеральная нотация
* Конструкторы объектов
* Ассоциативные массивы

***Оператор new***

var MyObject = new Object();

//Переменные

MyObject.id = 5; //Число

//Функции

MyObject.getName = function()

{

return this.name;

}

***Литеральная нотация***

MyObject = {

id : 1,

name : "Sample",

boolval : true,

getName : function()

{

return this.name;

}}

Минус этих способов заключается в том, что они позволяют работать только с одним вновь созданным объектом.

***Конструкторы объектов***

Конструкторы объектов — это мощное средство для создания объектов, которые можно использовать неоднократно. Создавая различные экземпляры объекта, можно работать отдельно с каждым, не боясь того, что, изменяя свойства одного экземпляра, мы затронем свойства другого.

Как и в ООП, у MyObject могут быть методы и различные свойства. Свойствам можно присвоить значения по умолчанию, либо значения, переданные пользователем в конструкторе объекта.

function MyObject(id, name)

{

//Значения переданные пользователем

this.\_id = id;

this.\_name = name;

//Значение по умолчанию

this.defaultvalue = "MyDefaultValue";

}

***Ассоциативные массивы***

Подобный метод будет полезен для упорядочивания большого числа однотипных объектов.

var MyObject = new Number();

MyObject["id"] = 5;

MyObject["name"] = "SampleName";

Для обхода таких объектов можно использовать такой цикл:

for (MyElement in MyObject)

{

//Код обхода

//В MyElement - идентификатор записи

//В MyObject[MyElement] - содержание записи

}

Prototyping, \_\_proto\_\_, inheritance, constructor, levels of visibility

***What is Prototype*?**

Prototype – свойство объекта (изначально пустое), в которое можно добавлять свойства и методы как в обычный объект.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10 | var myObject = function(name){      this.name = name;      return this;  };    console.log(typeof myObject.prototype); // object    myObject.prototype.getName = function(){      return this.name;  }; |

## The Secret Link

*Every object within JavaScript has a “secret” property.*

Каждый объект в javascript по умолчанию имеет свойство ***\_\_proto\_\_***, за счет которого реализуется цепочка наследования.

Тем не менее, использовать это свойство в коде не стоит, так как его поддерживают не все браузеры

Свойство \_\_proto\_\_ не стоит путать со свойством prototype

Когда мы определяем myObject, у этого объекта создается свойство \_\_proto\_\_, которое будет ссылаться на Function.prototype.

Далее, когда мы захотим получить доступ к myObject.length , сначала оно будет искаться в самом объекте, и когда не найдет, начнет подниматься по цепочке прототипов, следуя ссылке, указанной в свойстве \_\_proto\_\_

console.log(myObject instanceof Function); // true

console.log(myObject === Function); // false

Когда создается экземпляр объекта, свойство \_\_proto\_\_ начинает ссылаться на прототип конструктора

То есть, когда мы создаем новую функцию, ее \_\_proto\_\_ ссылается на Function.prototype, а когда объект создается с помощью оператора new – на Object.prototype

|  |  |
| --- | --- |
|  | console.log(myObject.\_\_proto\_\_ === Function.prototype) // true |
|  |  |
|  |  |

var myInstance = new myObject(“foo”);

console.log(myInstance.\_\_proto\_\_ === myObject.prototype); // true

## Constructor

Свойство, которое имеет любой объект, возвращает ссылку на функцию-конструктор, которая создала данный объект. Возвращается именно ссылка, а не строка с названием

var o = {};

o.constructor === Object; // true

var a = [];

a.constructor === Array; // true

var n = new Number(3);

n.constructor === Number; // true

## How to Use Prototype

## Свойства и методы, помещенные в prototype, будут доступны всем экземплярам, так же, как если б это были их собственные свойства. Это значительно помогает уменьшить память, необходимую приложению.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19 | // define the GameObject constructor function  var GameObject = function(width, height) {      this.x = Math.floor((Math.random() \* myCanvasWidth) + 1);      this.y = Math.floor((Math.random() \* myCanvasHeight) + 1);      this.width = width;      this.height = height;      return this;  };    // (re)define the GameObject prototype object  GameObject.prototype = {      x: 0,      y: 0,      width: 5,      width: 5,      draw: function() {          myCanvasContext.fillRect(this.x, this.y, this.width, this.height);      }  }; |

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var x = 100,  arrayOfGameObjects = [];    do {      arrayOfGameObjects.push(new GameObject(10, 10));  } while(x--); |

|  |  |
| --- | --- |
| 1  2  3  4  5 | var GameLoop = function() {      for(gameObject in arrayOfGameObjects) {          gameObject.draw();      }  }; |

**Prototype is a Live Object**

Можно модифицировать методы в prototype, они также обновятся в уже существующих экземплярах

**Updating Native Objects Prototypes**

Можно добавлять свойства и методы непосредственно в базовые объектные типы, тогда созданные методы и свойства будут доступны во всех экземплярах

|  |  |
| --- | --- |
| 1  2  3 | String.prototype.trim = function() {      return this.replace(/^\s+|\s+$/g, ‘’);  }; |

|  |  |
| --- | --- |
| 1 | “ foo bar   “.trim(); // “foo bar” |

Тем не менее, не рекомендуется так делать, потому что со временем может появиться нативная реализация, которая перекроется нашей реализацией.

Поэтому, если и добавлять что-то в базовые прототипы, необходимо проверять существование метода, который планируется создать

Links

<http://yehudakatz.com/2011/08/12/understanding-prototypes-in-javascript/>

<http://sporto.github.io/blog/2013/02/22/a-plain-english-guide-to-javascript-prototypes/>

<http://javascriptissexy.com/javascript-prototype-in-plain-detailed-language/>

<http://karaboz.ru/?p=9>

<http://code.tutsplus.com/tutorials/prototypes-in-javascript--net-24949>

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Object/constructor