Основы JS (5)

Вернемся к объектам

Лекция посвящена особенностям работы с объектами, о которых шел разговор на прошлом занятии. Если что-то сложно – стоит обратиться к прошлой лекции

Здравствуй "this"!

- Для доступа к текущему объекту из метода используется ключевое слово this.
- Мы помним наш пример

```
cat = {hunger: 100}
cat.feed = function () {
cat.hunger-=20;//cat ссылается на текущий
   объект
alert('Myppp....')
}
console.log(cat.hunger); // 100
cat.feed(); //Myppp....
console.log(cat.hunger); // 80
```

this this this

- Мы помним наш пример

```
cat = {hunger: 100}
cat.feed = function () {
this.hunger-=20;//Универсальнее использовать
//this, чтобы не привязываться к имени
   объекта
alert('Myppp....')
}
console.log(cat.hunger); // 100
cat.feed(); //Myppp.... Работает без ошибок
console.log(cat.hunger); // 80
```

Когда без this не обойтись?

Объект может быть создан при помощи любой функции – конструктора.

!!!В ней мы не знаем имени будущего объекта!!!

Напишем функцию-конструктор котов

```
function cat(catname){
   this.catname=catname;//Публичное поле
   this.hunger = 100;
   this.feed = function(){
    this.hunger-=20;
   alert('Myppp....');
   }
}
```

Создадим кота

- Конструктором становится любая функция, вызванная через new.

```
mycat= new cat('Барсик');
```

К созданному объекту обращения идут аналогично:

```
console.log(mycat.hunger);
mycat.feed();
console.log(mycat.hunger);
```

Барсик покушал, Бабарсик – нет.

```
mycat= new cat('Барсик');
mysecondcat= new cat('Бабарсик');
console.log(mycat.hunger); // 100
console.log(mysecondcat.hunger); // 100
mycat.feed();
console.log(mycat.hunger); // 80
console.log(mysecondcat.hunger); // 100
```

Оставим кота и создадим бойца

```
function warrior(myname,startx,starty){
this.x=startx;
this.y=starty;
this.up = function (){this.y++;}
this.down = function (){this.y--;}
this.left = function (){this.x--;}
this.right = function (){this.x++;}
}
valera= new warrior('Валера',5,5);
valera.up();//5 6
valera.up();//5 7
valera.left();//4 7
console.log('x='+valera.x+' y='+valera.y);
```

Теперь давайте научим бойца

```
//Сделаем один метод для передвижения
function warrior(myname,startx,starty){
    this.x=startx;
    this.y=starty;
    this.up = function (){this.y++;}
    this.down = function (){this.y--;}
    this.left = function (){this.x--;}
    this.right = function (){this.x++;}
    this.move = function(command){
        this[command]();//Работает, не
рекондовано
    }
}
```

Воспользуемся методом move()

```
valera= new warrior('Валера',5,5);
valera.move('up');//5 6
valera.move('up');//5 7
valera.move('left');//4 7
console.log('x='+valera.x+' y='+valera.y);
```

Но прежние методы работают...

valera.up(),valera.down(),valera.left(),valera.right()

Они по прежнему работают! Закроем их для пользователя...

```
up = function (){this.y++;}
down = function (){this.y--;}
left = function (){this.x--;}
right = function (){this.x++;}
this.move = function(command){
eval( command )();//не рекомендуется
//аналогичен up(), где command='up'
}
```

Проверим

```
valera= new warrior('Валера',5,5);
valera.move('up');//5 6 ?
valera.move('up');//5 7 ?
valera.move('left');//4 7 ?
console.log('x='+valera.x+' y='+valera.y); // 5 5
```

Всё сломалось. Что делать?

```
up = function (){this.y++;}
down = function (){this.y--;}
left = function (){this.x--;}
right = function (){this.x++;}
```

Доступ к объекту из внутреннего метода. call(this)

```
function warrior(myname,startx,starty){
    this.x=startx;
    this.y=starty;
    up = function (){this.y++;}
    down = function (){this.y--;}
    left = function (){this.x--;}
  right = function (){this.x++;}
this.move = function(command){
eval(command).call(this);//аналогично
up.call(this) для command= 'up'
    }
}
```

```
valera= new warrior('Валера',5,5);
valera.move('up');//5 6
valera.move('up');//5 7
valera.move('left');//4 7
console.log('x='+valera.x+' y='+valera.y);
```

Ура! Починили!

.bind(this)

```
function warrior(myname,startx,starty){
    this.x=startx;
    this.y=starty;
    up = function (){this.y++;}.bind(this);
    down = function (){this.y--;}.bind(this);
    left = function (){this.x--;}.bind(this);
  right = function (){this.x++;}.bind(this);
this.move = function(command){
eval(command)();//Вызов без call
    }
}
```

var self = this; self – имя переменной
Создаем замыкание!

```
function warrior(myname,startx,starty){
    this.x=startx;
    this.y=starty;
var self = this;
    up = function (){self.y++;};
    down = function (){self.y--;};
    left = function (){self.x--;};
  right = function (){self.x++;};
this.move = function(command){
eval(command)();//Работает, не рекондовано
    }
}
```

Мы "забыли" о самом главном

Свойства x и y – публичные и доступны извне

Параметры вызова (myname,startx,starty) являются приватными и доступны только внутри объекта.

Итог – мы можем телепортировать бойца по полю через

```
valera.x=1000;
valera.y=500;
console.log('x='+valera.x+' y='+valera.y);
```

Сделаем координаты приватными

```
function warrior(myname,startx,starty){
    up = function (){starty++;};
    down = function (){starty--;};
    left = function (){startx--;};
    right = function (){startx++;};
    this.move = function(command){
    eval(command)();//Работает, не рекондовано
    }
}
valera= new warrior('Валера',5,5);
valera.move('up');//5 6
valera.move('left');//4 6
console.log('x='+valera.x+' y='+valera.y);//? Почему
```

Валера, сообщите координаты!

```
function warrior(myname,startx,starty){
    up = function (){starty++;};
    down = function (){starty--;};
    left = function (){startx--;};
    right = function (){startx++;};
    this.move = function(command){
    eval(command)();//Работает, не рекондовано
    }
    this.report= function (){
        console.log(myname+' докладывает: Мои
            координаты - x='+startx+' y='+starty);
    }
}
```

Вот теперь всё как нужно!

```
valera= new warrior('Валера',5,5);
valera.move('up');//5 6
valera.move('up');//5 7
valera.move('left');//4 7
valera.report();
//Валера докладывает: Мои координаты - x=4 y=7
//Как сделать, чтобы код ниже делал то же?
valera= new warrior('Валера',5,5);
valera.move('up').move('up').move('left');//4 7
valera.report();
```

Getter and Setter

Для лучшего контроля изменения приватных полей созданы два брата:

- Getter – Получает/высчитывает значение
- Setter – Устанавливает/изменяет значение

Валера, вот твой автомат

Валера прошел стройподготовку и самое время снарядить его автоматом

```
function warrior(myname,weapon,maxbullets){
var bullets; // Текущее число патронов
}

valera = new warrior('Валера','Автомат',300);
```

Держи патроны!

```
function warrior(myname,weapon,maxbullets){
var bullets=0; // Текущее число патронов
this.getbullets = function(amount){//это setter =)
    if (amount<=0) {alert('Дай больше 0!');return;}
    if (amount+bullets>maxbullets) {
        console.log('Взял ' +(maxbullets-bullets)+ '
            патронов из '+amount);
        bullets=maxbullets;
        } else{
        bullets+=amount;
        console.log('Взял '+bullets+' патронов');}
    }
}
```

Валера, отчитайся

```
this.report = function (){
console.log('У меня '+bullets+' патронов из
'+maxbullets);
}

valera = new warrior('Валера','Автомат',300);
valera.report();
valera.getbullets(50);
valera.getbullets(270);
valera.report();
```

Задачи

1. Сделайте единый getter-setter:
getbullets() - getter сообщает количество патронов
getbullets(amount) – берет патроны.
(Код можно использовать из прошлого примера)
1.* Расширьте класс воина, чтобы Валера мог:
- перезаряжать автомат(если есть чем)
- стрелять, пока есть патроны
- говорить сколько у него осталось абойм (floor/ceil) на выбор

Задачи

2. Взять объект "Сматфон" у которого заданы поля: имя, память(hdd), память(ram) и список приложений (имя, hdd, ram) из 4 лекции.
- Переписать все его поля на приватные.
- дать возможность установить карту памяти.
- Сделать “перенос” приложений на карту памяти и обратно.
* - Возможность достать карту памяти с сохранением работоспособности приложений на внутренней памяти

Задачи

3. Валера в лабиринте
Написать класс labirintwarrior,
Карта, старт и финиш приходят в конструкторе.
- боец может перемещаться и бить мечем перед собой(направление движения/поворота)
- боец может отобразить карту в радиусе N=5 от себя (* с отображением в HTML)
Карта состоит из 3 типов клеток:
- проходимые(корридоры)
- разбиваемые(двери/коробки)
- непроходимые(стены)
Нужно пройти лабиринт

Реализовать объект "автомобиль", для которого прописать не менее 10 методов.
Переменные(объем топлива, масла, температура, вместимость багажника или другие – приватные)
Хотя бы 3 метода должны вызывать приватные методы внутри себя.
* Сделать движение автомобиля по "карте".
- Поворот автомобиля на 90 градусов.
- Реализовать инерцию с торможением
- Реализовать торможение