Урок 4

Объекты в JavaScript.

Понятие объектов, реализация объектов в JS. Работа с объектами.

Введение

На предыдущих занятиях мы программировали на базе простых данных и структур. Такой тип программирования называется процедурным. Но в JavaScript мы уже использовали совершенно чуждую для этого подхода сущность – объект. При этом, мы пока даже не догадывались, что используем объекты. Но на этом уроке мы научимся осознанно создавать и применять объекты. Вполне вероятно, что, привыкнув к ним, Вы уже не захотите снова писать процедурный код!

Объекты в JavaScript

В JS объекты занимают ключевую позицию в построении сложного кода. Это может быть управление моделью HTML-документа, организация хранения данных, упаковка библиотек JavaScript. Тема объектов пронизывает язык практически насквозь.

Главный секрет объектов JavaScript состоит в том, что они представляют собой коллекцию свойств. В качестве примера мы рассмотрим бы автомобиль. Он обладает следующими свойствами:

- Марка;
- Модель;
- Цвет;
- Количество пассажиров;
- Мощность двигателя;
- Год изготовления.

Разумеется, полный список свойств реального автомобиля не ограничивается вышеперечисленными пунктами. Но в программе зачастую все свойства не нужны, а требуются только некоторые из них. Поэтому давайте переведём то, что мы написали выше на JavaScript.
Для того, чтобы создать объект, нужно присвоить наши свойства некой переменной, чтобы впоследствии иметь возможность манипулировать объектом. Для этого после стандартного объявления переменной и символа «=» необходимо открыть фигурные скобки. Внутри них перечисляются все необходимые нам свойства.

```
var car = {
  make: "Audi",
  model: "A5",
  year: 2015,
  color: "red",
  passengers: 2,
  power: 225
};
```

Только что мы создали самый настоящий JS-объект с набором свойств. Мы тут же присвоили его переменной, делая возможным обращение к объекту в дальнейшем. С этого момента у нас есть возможность обращаться к нашему объекту, читать значения его свойств, редактировать их, а также добавлять новые свойства или удалить их.

Итак, при создании объекта:

1. Заключайте определение объекта в фигурные скобки.
2. Имя свойства отделяется от значения двоеточием.
3. Имя свойства может быть произвольной строкой.

4. Объект не может содержать два свойства с одинаковыми именами.
5. Пары «имя/значение» свойств разделяются запятыми.
6. После значения последнего свойства запятая не ставится.

Объекты в JavaScript

Итак, мы начали работать с объектами, уходя тем самым от процедурного подхода к программированию. Теперь решаемая задача будет рассматриваться не как набор переменных, условий, циклов и функций. В объектно-ориентированном программировании (ООП) решение задачи происходит в контексте объектов. Они обладают неким состоянием (значения переменных у конкретного объекта) и поведением (функции, которые доступны отдельно взятому объекту).

Какие преимущества приносит такой подход? Мы начинаем мыслить не в искусственно выдуманных сущностях переменных и функций, а переходим на реальный и более высокий уровень представления сущностей. Ведь в повседневной жизни мы как раз и реализуем концепцию объектов. Допустим, при приготовлении кофе Вы не бежите выращивать кофейное дерево, собирать зерна и так далее. Вы берете объект Кофемашина и выполняете у него метод Сварить кофе (Арабика). Более подробно концепция ООП разбирается на курсе JavaScript Level 2. Пока нам достаточно нашего уровня знаний для применения объектов в JS.

Объекты позволяют скрывают сложную структуру данных, давая разработчикам возможность работать на высоком уровне кода и не тратить время на технические нюансы. В примере с кофе-машиной Вам не нужно знать, как происходит нагрев воды или подача её под давлением на перемолотый кофе.

Примечание [1]: скрывать

Свойства объектов

Таким образом, у нас есть объект с упакованными внутри него свойствами. Для того, чтобы начать работать со свойством объекта, нужно указать имя объекта, поставить точку, после чего указать имя свойства. Такой синтаксис (с точечной записью), выглядит так

```
var car = {
  make: "Audi",
  model: "A5",
  year: 2015,
  color: "red",
  passengers: 2,
  power: 225
};
console.log(car.model);
car.power = 300;
```

В любой момент после инициации объекта его можно дополнить новыми свойствами. Для этого нужно указать новое свойство и присвоить ему значение

```
car.odometer = 100;
```

Для удаления свойств необходимо применить ключевое слово delete. Оно уничтожает не только значение свойства, но и его само. При попытке обращения к удалённому свойству результат будет равен undefined.

```
delete car.odometer;
```

Сам delete возвращает true при успешном удалении свойства. false вернётся только в случае, когда свойство не было удалено (например, если свойство входит в защищенный объект, принадлежащий браузеру).

Можно создать объект без свойств или же объект с сотнями свойств. Ограничений тут нет.

Хранение объектов

В отличие от простых переменных (строк, чисел), которые мы непосредственно помещаем в переменную, объекты хранятся несколько иначе. Когда мы присваиваем переменной объект, то, по сути, мы помещаем в неё ссылку на объект. Т.е. переменная будет содержать не сам объект, а указатель на его область в памяти. И в JS мы не знаем, как именно выглядит этот указатель, но, так или иначе, он указывает на наш объект.

В случае процедурного подхода аргументы передаются функциям по значению. Т.е. мы создаём копию текущей переменной, что-то с этой копией делаем, но исходная переменная при этом не меняется. Те же правила работают и для объектов, но ведут они себя несколько иначе. Поскольку в переменной хранится ссылка на объект, а не сам объект, при рассмотрении передачи по значению в параметре передаётся копия ссылки, которая, по сути, указывает на исходный объект. Таким образом, в отличие от примитивов, при изменении свойства объекта в функции изменяется свойство исходного объекта. Следовательно, все изменения, вносимые в объект внутри функции, продолжат действовать и после завершения функции.

```
var car = {
  make: "Audi",
  model: "A5",
  year: 2015,
  color: "red",
  passengers: 2,
  power: 225,
  odometer: 0
};
function haveRoadTrip(my_car, distance){
  my_car.odometer += distance;
}
haveRoadTrip(car, 150);
console.log(car.odometer);
```

Методы объектов

Помимо свойств, которые описывают состояние объекта, в JS у объектов есть методы, определяющие поведение. Объекты активны и могут выполнять различные операции. Автомобиль не стоит на месте – он передвигается, включает фары и т.д. Таким образом, объект car также должен уметь совершать эти действия.

Метод можно добавить непосредственно в объект. Это делается следующим образом.

```
var car = {
  make: "Audi",
  model: "A5",
  year: 2015,
  color: "red",
  passengers: 2,
  power: 225,
  odometer: 0,
  startEngine: function() {
    console.log("Engine started");
  }
};
```

Как видите, объявление метода представляет собой простое присвоение функции свойству. При этом мы не указываем имя функции, а ставим ключевое слово function, после которого уже описываем само поведение метода. Имя метода совпадает с именем свойства.
При вызове метода startEngine используется точечная запись, только вместо свойства мы пишем имя функции и круглые скобки, внутри которых при необходимости можем перечислять аргументы.

```
car.startEngine();
```

Пока мы оперируем простым выводом строк в консоль. Но давайте усложним наш объект и добавим к нему функцию haveRoadTrip. Она изменится – теперь нам не надо передавать в неё объект автомобиля. Если рассудить логически, то машина никуда не поедет с заглушенным двигателем. Поэтому нам понадобятся:

- Булевое свойство для хранения состояния двигателя (запущен или нет).
- Условная проверка в методе haveRoadTrip, которая удостоверяется, что двигатель запущен, прежде чем вы сможете вести машину.

```
var car = {
  make: "Audi",
  model: "A5",
  year: 2015,
  color: "red",
  passengers: 2,
  power: 225,
  odometer: 0,
  engineIsStarted: false,
  startEngine: function() {
    this.engineIsStarted = true;
  },
  stopEngine: function() {
    this.engineIsStarted = false;
  },
  haveRoadTrip: function(distance) {
    if (this.engineIsStarted) {
      this.odometer += distance;
    } else {
      alert("Сначала запустите двигатель!");
    }
  }
};
```

Чтобы запустить двигатель, мы могли бы убрать метод startEngine, а вместо него использовать простое изменение значения свойства. Но если подумать, то мы поймём, что запуск двигателя может быть сложнее. Что, если мы нарастим туда проверку заряда аккумулятора или снятие с сигнализации? Каждый раз вызвать такой код будет неудобно. Лучше создать единый метод, который знает весь технический процесс запуска двигателя.

Вы, наверняка, заметили, что в коде появилось странное слово this. Что же оно обозначает?
В наших методах переменные engineIsStarted и odometer не являются локальными или глобальными. Они являются свойством объекта car. Именно для них в JavaScript существует ключевое слово this: оно обозначает текущий объект, с которым мы работаем.

Можно представлять this в качестве переменной, которая указывает на объект, метод которого был только что вызван. Таким образом, если вызвать метод startEngine объекта car, то this будет указывать на объект car. При вызове любого метода вы можете быть уверены, что this в теле метода будет указывать на объект, метод которого был вызван.
Таким образом мы видим, что:

1. Поведение влияет на состояние. Т.е. внутри методов объекта производится изменение значений его свойств.
2. Состояние влияет на поведение. Не совсем очевидная истина, но посмотрите на проверку на заведенный двигатель. В зависимости от значения true/false метод будет вести себя по-

разному.

Перебор значений

Как Вы помните из предыдущего урока, есть особый вид цикла – for..in. Он перебирает все свойства объекта в произвольном порядке. Таким образом, наш объект car может передать нам все свои свойства следующим образом:

```
for (var prop in car) {
   console.log(prop + ": " + chevy[prop]);
}
```

Цикл for in перебирает свойства объекта, которые последовательно присваиваются переменной prop. Также Вы уже могли заметить, что мы применили альтернативный способ обращения к свойствам объекта: запись с квадратными скобками. Она эквивалентна известной нам точечной записи, делает то же самое, но обладает чуть большей гибкостью.

Практикум. Квест

Что ж, раз мы умеем удобно хранить состояние, то можем вполне реализовать текстовый квест. Это разновидность компьютерных игр, в которых общение с игроком осуществляется посредством текстовой информации. Развитие этого жанра, в связи с низким требованием к ресурсам, началось практически вместе с появлением компьютерных игр (появились уже в 1975 году) и не прекратилось даже с появлением графических игр.

Мы реализуем нашу игру в консоли.

Алгоритм будет таким:

1. Браузер генерирует приглашает пользователя к игре.
2. Выводится текущее состояние.
3. Выводится предложение о ходе.
4. Пользователь вводит действие.
5. В зависимости от действия генерируется следующий шаг.

Домашнее задание

1. Написать функцию, преобразующую число в объект. Передавая на вход число от 0 до 999, мы должны получить на выходе объект, в котором в соответствующих свойствах описаны единицы, десятки и сотни. Например, для числа 245 мы должны получить следующий объект: {‘единицы’: 5, ‘десятки’: 4, ‘сотни’: 2}. Если число превышает 999, необходимо выдать соответствующее сообщение с помощью console.log и вернуть пустой объект.
2. Продолжаем работу с нашим интернет-магазином

   2.1. В прошлом ДЗ Вы реализовали корзину на базе массивов. Какими объектами можно заменить элементы данных массивов?

   2.2. Реализуйте такие объекты

   2.3. Перенесите функционал подсчета корзины на объектно-ориентированную базу
3. * Подумайте над глобальными сущностями. К примеру, сущность “Продукт” в интернет-магазине актуальна не только для корзины, но и для каталога. Стремиться нужно к тому, чтобы объект “Продукт” имел единую структуру для различных модулей нашего сайта, но в разных местах давал возможность вызывать разные методы.

Дополнительные материалы

1. https://habrahabr.ru/post/48542/ - работа с объектами в JS

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. «JavaScript. Подробное руководство» - Дэвид Флэнаган
2. «Изучаем программирование на JavaScript» - Эрик Фримен, Элизабет Робсон