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 — то есть по процедурному типу. Но в JavaScript уже использовали совершенно чуждую для этого подхода сущность — объект. Научимся осознанно создавать и применять объекты. Вполне вероятно, что, привыкнув к ним, вы больше не захотите 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.

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

- марка;
- модель;
- цвет;
- количество пассажиров;
- мощность двигателя;
- год выпуска.

Разумеется, полный список свойств реального автомобиля не ограничивается этими пунктами. Но в программе зачастую все свойства не нужны, а требуются только некоторые. Переведем написанное на JavaScript.

Чтобы создать объект, нужно присвоить эти свойства переменной — это позволит впоследствии манипулировать объектом. Для этого после стандартного объявления переменной и символа **=** необходимо открыть фигурные скобки. Внутри перечисляются все необходимые свойства.

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

Так создается JS-объект с набором свойств. Мы сразу присвоили его переменной, чтобы обращаться к нему в дальнейшем. Также можно читать значения его свойств, редактировать их, добавлять новые свойства или удалять их.

Рекомендации 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

Работать с объектами — значит уходить от процедурного подхода к программированию. Теперь задача будет рассматриваться не как набор переменных, условий, циклов и функций. В объектно-ориентированном программировании (ООП) решение задачи происходит в контексте объектов. Они обладают состоянием (значения переменных у конкретного объекта) и поведением (функции, которые доступны конкретному объекту).

Используя ООП, мы перестаем мыслить в выдуманных переменных и функциях, а переходим на реальный и более высокий уровень представления сущностей. Ведь в повседневной жизни как раз и реализуется концепция объектов.

Чтобы сварить кофе, мы не бежим выращивать кофейное дерево и собирать зерна, а берем объект «кофемашина» и выполняем у него метод «сварить кофе (Арабика)». Более подробно концепция ООП разбирается на продвинутом курсе JavaScript.

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

Свойства объектов

Есть объект с упакованными внутри него свойствами, и чтобы начать работать с ними, нужно указать имя объекта, поставить точку, а после написать 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;
```

Чтобы удалить свойство, необходимо применить ключевое слово **delete**. Оно уничтожает и само свойство, и его значение. При попытке обращения к удаленному свойству результатом будет

undefined.

```
delete car.odometer;
```

Сам **delete** возвращает **true** при успешном удалении свойства, а **false** вернется только в случае, когда свойство не было удалено. Например, если свойство входит в защищенный объект, принадлежащий браузеру.

Можно создать объект с сотнями свойств или без них — ограничений нет.

Хранение объектов

Объекты хранятся иначе, чем простые переменные (строки, числа), которые непосредственно помещаются в переменную. Когда переменной присваивается объект, то в нее помещается ссылка на него. Переменная будет содержать не сам объект, а указатель на его область в памяти. В JS мы не знаем, как именно выглядит этот указатель на объект.

При процедурном подходе аргументы передаются функциям по значению. Создается копия переменной, с ней выполняются действия, но исходная переменная при этом не меняется. Те же правила работают и для объектов, но ведут они себя иначе. Поскольку в переменной хранится ссылка на объект, а не он сам, при передаче по значению в параметре передается копия ссылки, которая указывает на исходный объект. В отличие от примитивов, при изменении свойства объекта в функции меняется свойство исходного объекта. Значит все изменения, вносимые в объект внутри функции, продолжат действовать и после ее завершения.

```
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 операции. Автомобиль не стоит на месте — он передвигается, включает фары. Таким образом, объект **car** также должен действовать.

Метод можно добавить непосредственно в объект:

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

Объявление метода представляет собой простое присвоение функции свойству. При этом не указывается имя функции, а ставится ключевое слово **function**, после которого уже описывается поведение метода. Имя метода совпадает с именем свойства.

При вызове метода **startEngine** используется точечная запись, только вместо свойства пишется имя функции и круглые скобки, внутри которых при необходимости перечисляются аргументы.

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

Пока мы оперируем простым выводом строк в консоль. Усложним объект и добавим к нему функцию **haveRoadTrip**. Она изменится — теперь не надо передавать в нее объект автомобиля. Если рассудить логически, машина не поедет с заглушенным двигателем. Поэтому 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, можно было бы убрать метод **startEngine**, а вместо него просто изменять значения свойства. Но запуск двигателя может быть сложнее: можно добавить 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.

Рассмотрим слово **this**, которое появилось в коде.

В методах переменные **engineIsStarted** и **odometer** не являются локальными или глобальными — это свойства объекта **car**. Именно для них в JavaScript существует ключевое слово **this**: оно обозначает текущий объект, с которым ведется работа.

Можно представлять **this** как переменную, которая указывает на объект, метод которого был только что вызван. Если вызвать метод **startEngine** объекта **car**, то **this** будет указывать на объект **car**. При вызове любого метода можно быть уверенными, что **this** в теле метода будет указывать на объект, метод которого был вызван.

Подведем итог:

1. Поведение влияет на состояние. Внутри методов объекта изменяются значения его свойств.
2. Состояние влияет на поведение. Эта истина не совсем очевидная, но вспомним проверку на заведенный двигатель. В зависимости от значения **true**/**false** метод будет вести себя по-разному.

Перебор значений

Особый вид цикла **for..in** перебирает все свойства объекта в произвольном порядке. Объект **car** может передать 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**.

Мы применили альтернативный способ обращения к свойствам объекта: запись с квадратными скобками. Она эквивалентна точечной записи, но более гибкая.

Практикум. Квест

Раз мы умеем удобно хранить состояние, то можем реализовать текстовый квест, где будем общаться с игроком посредством текста. Этот жанр требует мало ресурсов, поэтому развивается с момента появления компьютерных игр — с 1975 года.

Реализуем игру в консоли. Алгоритм будет таким:

1. Браузер 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.

Практическое задание

1. Написать функцию, преобразующую число в объект. Передавая на вход число от 0 до 999, надо получить на выходе объект, в котором в соответствующих свойствах описаны единицы, десятки и сотни. Например, для числа 245 надо 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. Продолжить работу с интернет-магазином:
    a. В прошлом домашнем задании вы реализовали корзину на базе массивов. Какими объектами можно заменить их элементы?
    b. Реализуйте такие объекты.
    c. Перенести функционал подсчета корзины на объектно-ориентированную базу.
3. * Подумать 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 сайта, но в разных местах давал возможность вызывать разные методы.

Дополнительные материалы

1. Работа с объектами в JavaScript: теория и практика.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

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