15 - Объекты.

1. Введение

Объект является фундаментальным типом данных в языке JavaScript. Объект – это составное значение: он объединяет в себе набор значений (простых значений или других объектов) и позволяет сохранять и извлекать эти значения по именам. Объект является неупорядоченной коллекцией свойств, каждое из которых имеет имя и значение. Помимо собственных свойств объекты в языке JavaScript могут также наследовать свойства от других объектов, известных под названием «прототипы». Методы объекта – это типичные представители унаследованных свойств, а «наследование через прототипы» является ключевой особенностью языка JavaScript.

Объекты в языке JavaScript являются динамическими – обычно они позволяют добавлять и удалять свойства – но они могут использоваться также для имитации статических объектов и «структур», которые имеются в языках программирования со статической системой типов.

Любое значение в языке JavaScript, не являющееся строкой, числом, true, false, null или undefined, является объектом. Как вы помните, объекты являются изменяемыми значениями и операции с ними выполняются по ссылке, а не по значению. Если переменная x ссылается на объект, и выполняется инструкция var y = x;, в переменную y будет записана ссылка на тот же самый объект, а не его копия. Любые изменения, выполняемые в объекте с помощью переменной y, будут также отражаться на переменной x.

Наиболее типичными операциями с объектами являются создание объектов, назначение, получение, удаление, проверка и перечисление их свойств.

Свойство имеет имя и значение. Именем свойства может быть любая строка, включая и пустую строку, но объект не может иметь два свойства с одинаковыми именами. Значением свойства может быть любое значение, допустимое в языке JavaScript. В дополнение к именам и значениям каждое свойство имеет ряд ассоциированных с ним значений, которые называют **атрибутами свойства**:

• Атрибут writable определяет доступность значения свойства для записи.
• Атрибут enumerable определяет доступность имени свойства для перечисления в цикле for/in.
• Атрибут configurable определяет возможность настройки, т. е. удаления свойства и изменения его атрибутов.

В дополнение к свойствам каждый объект имеет три атрибута объекта:
• Атрибут prototype содержит ссылку на другой объект, от которого наследуются свойства.
• Атрибут class содержит строку с именем класса объекта и определяет тип объекта.
• Флаг extensible (в ECMAScript 5) указывает на возможность добавления новых свойств в объект.

Наконец, ниже приводится описание некоторых терминов, которые помогут нам различать три обширные категории объектов в языке JavaScript и два типа свойств:

• Объект базового языка – это объект, определяемый спецификацией ECMAScript. Массивы, функции, даты и регулярные выражения (например) являются объектами базового языка.

• Объект среды выполнения – это объект, определяемый средой выполнения (такой как веб-броузер), куда встроен интерпретатор JavaScript.

• Пользовательский объект – любой объект, созданный в результате выполнения программного кода JavaScript.

• Собственное свойство – это свойство, определяемое непосредственно в данном объекте.

• Унаследованное свойство – это свойство, определяемое прототипом объекта.

2. Создание объектов

Литералы объектов. Самый простой способ создать объект заключается во включении в программу литерала объекта. Литерал объекта – это заключенный в фигурные скобки список свойств (пар имя/значение), разделенных запятыми. Именем свойства может быть идентификатор или строковый литерал (допускается использовать пустую строку). Значением свойства может быть любое выражение, допустимое в JavaScript, – значение выражения (это может быть простое значение или объект) станет значением свойства. Ниже приводится несколько примеров создания объектов:

```
var empty = {}; // Объект без свойств
var point = { x:0, y:0 }; // Два свойства
var point2 = { x:point.x, y:point.y+1 }; // Более сложные значения
var book = {
    "main title": "JavaScript", // Имена свойств с пробелами
    'sub-title': "The Definitive Guide", // и дефисами, поэтому используются

    // строковые литералы
    "for": "all audiences", // for - зарезервированное слово,
    // поэтому в кавычках
    author: { // Значением этого свойства является
        firstname: "David", // объект. Обратите внимание, что
        surname: "Flanagan" // имена этих свойств без кавычек.
    }
};
```

Создание объектов с помощью оператора new. Оператор new создает и инициализирует новый объект. За этим оператором должно следовать имя функции. Функция, используемая таким способом, называется конструктором и служит для инициализации вновь созданного объекта. Базовый JavaScript включает множество встроенных конструкторов для создания объектов базового языка. Например:

```
var o = new Object(); // Создать новый пустой объект: то же, что и {}.
```

```
var a = new Array(); // Создать пустой массив: то же, что и [].
var d = new Date(); // Создать объект Date, представляющий текущее время
var r = new RegExp("js"); // Создать объект RegExp для операций
// сопоставления с шаблоном.
```

Помимо этих встроенных конструкторов имеется возможность определять свои собственные функции-конструкторы для инициализации вновь создаваемых объектов.

3. Прототипы

Каждый объект в языке JavaScript имеет второй объект (или null, но значительно реже), ассоциированный с ним. Этот второй объект называется прототипом, и первый объект наследует от прототипа его свойства. Все объекты, созданные с помощью литералов объектов, имеют один и тот же объект-прототип, на который в программе JavaScript можно сослаться так: Object.prototype. Объекты, созданные с помощью ключевого слова new и вызова конструктора, в качестве прототипа получают значение свойства prototype функции-конструктора. Поэтому объект, созданный выражением new Object(), наследует свойства объекта Object.prototype, как если бы он был создан с помощью литерала в фигурных скобках {}. Аналогично прототипом объекта, созданного выражением new Array(), является Array.prototype, а прототипом объекта, созданного выражением new Date(), является Date.prototype.

Object.prototype – один из немногих объектов, которые не имеют прототипа: у него нет унаследованных свойств. Другие объекты-прототипы являются самыми обычными объектами, имеющими собственные прототипы. Все встроенные конструкторы (и большинство пользовательских конструкторов) наследуют прототип Object.prototype. Например, Date.prototype наследует свойства от Object.prototype, поэтому объект Date, созданный выражением new Date(), наследует свойства от обоих прототипов, Date.prototype и Object.prototype. Такая связанная последовательность объектов-прототипов называется цепочкой прототипов.

4. Получение и изменение свойств

Получить значение свойства можно с помощью операторов точки (.) и квадратных скобок ([]). Слева от оператора должно находиться выражение, возвращающее объект. При использовании оператора точки справа должен находиться простой идентификатор, соответствующий имени свойства. При использовании квадратных скобок в квадратных скобках должно указываться выражение, возвращающее строку, содержащую имя требуемого свойства:

```
var author = book.author; // Получить свойство "author" объекта book.
var name = author.surname // Получить свойство "surname" объекта author.
var title = book["main title"] // Получить свойство "main title" объекта book.
```

Чтобы создать новое свойство или изменить значение существующего свойства, также используются операторы точки и квадратные скобки, как в операциях чтения

значений свойств, но само выражение помещается уже слева от оператора присваивания:

```
book.edition = 6; // Создать свойство "edition" объекта book.
book["main title"] = "ECMAScript"; // Изменить значение свойства "main title".
```

5. Объекты как ассоциативные массивы

Следующие два выражения возвращают одно и то же значение:

```
object.property
object["property"]
```

Вторая форма записи, с использованием квадратных скобок и строки, выглядит как обращение к элементу массива, но массива, который индексируется строками, а не числами. Такого рода массивы **называются ассоциативными массивами**. Объекты в языке JavaScript являются ассоциативными массивами, и в этом разделе объясняется, почему это так важно.

В C, C++, Java и других языках программирования со строгим контролем типов объект может иметь только фиксированное число свойств, а имена этих свойств должны определяться заранее. Поскольку JavaScript относится к языкам программирования со слабым контролем типов, данное правило в нем не действует: программы могут создавать любое количество свойств в любых объектах. Однако при использовании для обращения к свойству оператора точка (.) имя свойства определяется идентификатором. **Идентификаторы должны вводиться в тексте программы буквально – это не тип данных, поэтому в программе невозможно реализовать вычисление идентификаторов.**

Напротив, когда для обращения к свойствам объекта используется форма записи с квадратными скобками ([]), имя свойства определяется строкой. Строки в языке JavaScript являются типом данных, поэтому они могут создаваться и изменяться в ходе выполнения программы. Благодаря этому, например, в языке JavaScript имеется возможность писать такой программный код:

```
var addr = "";
for(i = 0; i < 4; i++)
      addr += customer["address" + i] + '\n';
```

Этот фрагмент читает и объединяет в одну строку значения свойств address0, address1, address2 и address3 объекта customer.

Этот короткий пример демонстрирует гибкость использования формы записи с квадратными скобками и строковыми выражениями для доступа к свойствам объекта.

6. Наследование

Объекты в языке JavaScript обладают множеством «собственных свойств» и могут также наследовать множество свойств от объекта-прототипа. Чтобы разобраться в этом, необходимо внимательно изучить механизм доступа к свойствам.

Предположим, что программа обращается к свойству x объекта o. Если объект o не имеет собственного свойства с таким именем, выполняется попытка отыскать свойство x в прототипе объекта o. Если объект-прототип не имеет собственного свойства

с этим именем, но имеет свой прототип, выполняется попытка отыскать свойство в прототипе прототипа. Так продолжается до тех пор, пока не будет найдено свойство x или пока не будет достигнут объект, не имеющий прототипа. Как видите, атрибут prototype объекта создает цепочку, или связанный список объектов, от которых наследуются свойства.

Теперь предположим, что программа присваивает некоторое значение свойству x объекта o. Если объект o уже имеет собственное свойство (не унаследованное) с именем x, то операция присваивания просто изменит значение существующего свойства. В противном случае в объекте o будет создано новое свойство с именем x.

Если прежде объект o наследовал свойство x, унаследованное свойство теперь окажется скрыто вновь созданным собственным свойством с тем же именем.

Операция присваивания значения свойству проверит наличие этого свойства в цепочке прототипов, чтобы убедиться в допустимости присваивания. Например, если объект o наследует свойство x, доступное только для чтения, то присваивание выполняться не будет. Однако если присваивание допустимо, всегда создается или изменяется свойство в оригинальном объекте и никогда в цепочке прототипов. **Тот факт, что механизм наследования действует при чтении свойств, но не действует при записи новых значений, является ключевой особенностью языка JavaScript, потому что она позволяет выборочно переопределять унаследованные свойства.**

7. Ошибки доступа к свойствам

Попытка обращения к несуществующему свойству не считается ошибкой. Если свойство x не будет найдено среди собственных или унаследованных свойств объекта o, выражение обращения к свойству o.x вернет значение undefined.

```
book.subtitle; // => undefined: свойство отсутствует
```

Однако попытка обратиться к свойству несуществующего объекта считается ошибкой. Значения null и undefined не имеют свойств, и попытки обратиться к свойствам этих значений считаются ошибкой. Продолжим пример, приведенный выше:

```
// Возбудит исключение TypeError. undefined не имеет свойства length
var len = book.subtitle.length;
```

Если нет уверенности, что book и book.subtitle являются объектами (или ведут себя подобно объектам), нельзя использовать выражение book.subtitle.length, так как оно может возбудить исключение. Ниже демонстрируются два способа защиты против исключений подобного рода:

```
// Более наглядный и прямолинейный способ
var len = undefined;
if (book) {
      if (book.subtitle) len = book.subtitle.length;
}

// краткая и характерная для JavaScript альтернатива получения длины
```

```
// значения свойства subtitle
var len = book && book.subtitle && book.subtitle.length;
```

Чтобы понять, почему второе выражение позволяет предотвратить появление исключений TypeError, можете вернуться к описанию короткой схемы вычислений, используемой оператором &&. Разумеется, попытка установить значение свойства для значения null или undefined также вызывает исключение TypeError.

8. Удаление свойств

Оператор delete удаляет свойство из объекта. Его единственный операнд должен быть выражением обращения к свойству. Может показаться удивительным, но оператор delete не оказывает влияния на значение свойства – он оперирует самим свойством:

```
delete book.author; // Теперь объект book не имеет свойства author.
delete book["main title"]; // Теперь он не имеет свойства "main title".
```

Оператор delete удаляет только собственные свойства и не удаляет унаследованные. (Чтобы удалить унаследованное свойство, необходимо удалять его в объекте-прототипе, в котором оно определено. Такая операция затронет все объекты, наследующие этот прототип.)

Выражение delete возвращает значение true в случае успешного удаления свойства или когда операция удаления не привела к изменению объекта (например, при попытке удалить несуществующее свойство). Выражение delete также возвращает true, когда этому оператору передается выражение, не являющееся выражением обращения к свойству:

```
o = {x:1}; // o имеет собственное свойство x и наследует toString
delete o.x; // Удалит x и вернет true
delete o.x; // Ничего не сделает (x не существует) и вернет true
delete o.toString; // Ничего не сделает (toString не собственное свойство) и вернет true
delete 1; // Бессмысленно, но вернет true
```