Объекты

Объекты – это составной тип данных, они объединяют множество значений в единый модуль и позволяют сохранять и извлекать значения по их именам. Говоря другими словами, объекты – это неупорядоченные коллекции свойств, каждое из которых имеет свои имя и значение. Именованные значения, хранящиеся в объекте, могут быть данными элемента

Создание 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 слож-
ные значения

var book = {

"main title": "JavaScript", // Имена свойств с пробелами

'sub-title': "The Definitive Guide", // и дефисами, по-
этому используются

// строковые литералы

"for": "all audiences", // for - зарезервированное сло-
во, поэтому в кавычках

author: {

    // Значением этого свойства является объект

    // имена этих свойств без кавычек.

    firstname: "David",

    Обратите внимание, что
```

```
        surname: "Flanagan" } };
```

Литерал объекта – это выражение, которое создает и инициализирует новый объект всякий раз, когда производится вычисление этого выражения. Значение каждого свойства вычисляется заново, когда вычисляется значение литерала. Это означает, что с помощью единственного литерала объекта можно создать множество новых объектов, если этот литерал поместить в тело цикла или функции, которая будет вызываться многократно, и что значения свойств этих объектов могут отличаться друг от друга.

С помощью оператора new можно создать другую разновидность объектов. За этим оператором должно быть указано имя функции-конструктора, выполняющей инициализацию свойств объекта. Например:

```
var a = new Array( ); // Создать пустой массив

var d = new Date( ); // Создать объект с текущими временем и датой

var r = new RegExp("javascript", "i"); // Создать объект регулярного выражения
```

Продемонстрированные здесь функции Array(), Date() и RegExp() являются встроенными конструкторами базового языка JavaScript. Конструктор Object() создает пустой объект, как если бы использовался литерал {}. Существует возможность определять собственные конструкторы для инициализации вновь создаваемых объектов необходимым вам способом.

Свойства объектов

Обычно для доступа к значениям свойств объекта используется оператор . (точка). Значение в левой части оператора должно быть ссылкой на объект, к свойствам которого требуется получить доступ. Обычно это просто имя переменной, содержащей ссылку на объект, но это может быть любое допустимое в JavaScript выражение, являющееся объектом.

Значение в правой части оператора должно быть именем свойства. Это должен быть идентификатор, а не строка или выражение. Так, обратиться к свойству p объекта o можно посредством выражения o.p, а к свойству radius объекта circle – посредством выражения circle.radius. Свойства объекта работают как переменные: в них можно сохранять значения и считывать их. Например:

```
// Создаем объект. Сохраняем ссылку на него в переменной.
```

```
 var book = new Object();
// Устанавливаем свойство в объекте.
book.title = "JavaScript: полное руководство"
// Устанавливаем другие свойства. Обратите внимание на вложенные объекты.
book.chapter1 = new Object();
book.chapter1.title = "Введение в JavaScript";
book.chapter1.pages = 11;
book.chapter2 = { title: "Лексическая структура", pages: 6 };
// Читаем значения некоторых свойств из объекта.
alert("Заголовок: " + book.title )
```

Важно обратить внимание на один момент в этом примере – новое свойство объекта можно добавить, просто присвоив этому свойству значение. Если переменные должны объявляться с помощью ключевого слова var, то для свойств объекта такой необходимости (и возможности) нет. К тому же после создания свойства объекта (в результате присваивания) значение свойства можно будет изменить в любой момент простым присваиванием ему нового значения:

```
book.title = "JavaScript: Книга с носорогом"
```

Объекты как ассоциативные массивы

Как мы знаем, доступ к свойствам объекта осуществляется посредством оператора «точка». Доступ к свойствам объекта возможен также при помощи оператора [], который обычно применяется при работе с массивами. Таким образом, следующие два JavaScript выражения имеют одинаковое значение:

object.property

object["property"]

Важное различие между этими двумя синтаксисами, на которое следует обратить внимание, состоит в том, что в первом варианте имя свойства представляет собой идентификатор, а во втором – строку. Скоро мы узнаем, почему это так важно.

В Java, C, C++ и подобных языках со строгой типизацией объект может иметь только фиксированное число свойств, и имена этих свойств должны быть определены заранее. Поскольку JavaScript – слабо типизированный язык, к нему данное правило неприменимо; программа может создавать любое количество свойств в любом объекте.

Однако в случае использования оператора «точка» для доступа к свойству объекта имя свойства задается идентификатором. Идентификаторы должны быть частью текста JavaScript программы – они не являются типом данных и ими нельзя манипулировать из программы. В то же время при обращении к свойству объекта с помощью нотации массивов [] имя свойства задается в виде строки. Строки в JavaScript – это тип данных, поэтому они могут создаваться и изменяться во время работы программы. И поэтому в JavaScript можно, например, написать следующий код:

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

В этом фрагменте читаются и объединяются в одну строку свойства address0, add ress1, address2 и address3 объекта customer.

Конструкторы

Ранее демонстрировался порядок создания новых пустых объектов как с по мощью литерала {}, так и с помощью следующего выражения: new Object()

Кроме того, была продемонстрирована возможность создания объектов других типов примерно следующим образом:

```
var array = new Array(10);
var today = new Date( );
```

За оператором new должно быть указано имя функции-конструктора. Оператор создает новый пустой объект без каких-либо свойств, а затем вызывает функцию, передавая ей только что созданный объект в виде значения ключевого слова this.

Функция, применяемая совместно с оператором new, называется функцией-конструктором, или просто конструктором. Главная задача конструктора заключается в инициализации вновь созданного объекта –

установке всех его свойств, которые необходимо инициализировать до того, как объект сможет использоваться программой.

Чтобы определить собственный конструктор, достаточно написать функцию, добавляющую новые свойства к объекту, на который ссылается ключевое слово this. В следующем фрагменте приводится определение конструктора, с помощью которого затем создаются два новых объекта: // Определяем конструктор.

```
// Обратите внимание, как инициализируется объект с помощью "this".

function Rectangle(w, h)

{ this.width = w; this.height = h; }

// Вызываем конструктор для создания двух объектов Rectangle. Мы передаем ширину и высоту конструктору, чтобы можно было правильно проинициализировать оба новых объекта.

var rect1 = new Rectangle(2, 4); // rect1 = { width:2, height:4 };

var rect2 = new Rectangle(8.5, 11); // rect2 = { width:8.5, height:11 };
```

Обратите внимание на то, как конструктор использует свои аргументы для инициализации свойств объекта, на который ссылается ключевое слово this. Здесь мы определили класс объектов, просто создав соответствующую функцию-конструктор – все объекты, созданные с помощью конструктора Rectangle(), гарантированно будут иметь инициализированные свойства width и height. Это означает, что учитывая данное обстоятельство, можно организовать единообразную работу со всеми объектами класса Rectangle.

Прототипы и наследование

Когда функция вызывается таким способом, объект, посредством которого производится вызов, становится значением ключевого слова this. Предположим, что необходимо рассчитать площадь прямоугольника, представленного объектом Rectangle. Вот один из возможных способов:

```
function computeAreaOfRectangle(r) { return r.width * r.height; }
```

Эта функция прекрасно справляется с возложенными на нее задачами, но она не является объектно-ориентированной. Работая с объектом, лучше

всего вызывать методы этого объекта, а не передавать объекты посторонним функциям в качестве аргументов. Этот подход демонстрируется в следующем фрагменте:

```
// Создать объект Rectangle
var r = new Rectangle(8.5, 11);
// Добавить к объекту метод
r.area = function() { return this.width * this.height; }
// Теперь рассчитать площадь, вызвав метод объекта
var a = r.area();
```

Конечно же, не совсем удобно добавлять новый метод к объекту перед его использованием. Однако ситуацию можно улучшить, если инициализировать свойство area в функции-конструкторе. Вот как выглядит улучшенная реализация конструктора Rectangle():

```
function Rectangle(w, h) {
      this.width = w; this.height = h;
     this.area = function( ) { return this.width *
this.height; } }
```

С новой версией конструктора тот же самый алгоритм можно реализовать по другому:

```
 // Найти площадь листа бумаги
var r = new Rectangle(8.5, 11);
var a = r.area();
```

Такое решение выглядит гораздо лучше, но оно по-прежнему не является оптимальным. Каждый созданный прямоугольник будет иметь три свойства. Свойства width и height могут иметь уникальные значения для каждого прямоугольника, но свойство area каждого отдельно взятого объекта Rectangle всегда будет ссылаться на одну и ту же функцию (разумеется, это свойство можно изменить в процессе работы, но, как правило, предполагается, что методы объекта не должны меняться).

Однако и эту проблему можно решить. Оказывается, все объекты в JavaScript содержат внутреннюю ссылку на объект, известный как прототип. Любые свойства прототипа становятся свойствами другого объекта, для которого он является прототипом. То есть, говоря другими словами, любой объект в JavaScript на' следует свойства своего прототипа. В предыдущем

разделе было показано, как оператор new создает пустой объект и затем вызывает функциюконструктор. Но история на этом не заканчивается. После создания пустого объекта оператор new устанавливает в этом объекте ссылку на прототип. Прототипом объекта является значение свойства prototype функции конструктора.

Более понятно это можно объяснить на примере. Вот новая версия конструктора Rectangle():

```
// Функция конструктор инициализирует те свойства, которые могут иметь уникальные значения для каждого отдельного экземпляра.

 function Rectangle(w, h) { this.width = w; this.height = h; }

// Прототип объекта содержит методы и другие свойства, которые должны совместно использоваться всеми экземплярами этого класса.

Rectangle.prototype.area = function() { return this.width * this.height; }
```

Расширение встроенных типов

Не только классы, определенные пользователем, имеют объекты-прототипы. Встроенные классы, такие как String и Date, также имеют объекты прототипы, и вы можете присваивать им значения. Например, следующий фрагмент определяет новый метод, доступный всем объектам String:

```
// Возвращает true, если последним символом является значение аргумента c

String.prototype.endsWith = function(c) { return (c == this.charAt(this.length1)) }
```

Определив новый метод endsWith() в объекте-прототипе String, мы сможем обратиться к нему следующим образом:

```
var message = "hello world"; message.endsWith('h') // Возвращает false

message.endsWith('d') // Возвращает true
```

Практическое задание

Продемонстрировать несколько способов создания объектов, несколько способов доступа к их свойствам (на чтение и запись).

Создать конструктор для собственного объекта, добавить в него несколько свойств и методов, продемонстрировать работу с этими объектами.

Расширить встроенный тип (например, Array или Date) функцией, оперирующей данными расширяемого объекта (например, вывод среднего арифметического для массива, или подсчет количества секунд со дня вашего рождения для даты).