## 04 - Циклы, инструкции перехода

### 1. Циклы

Чтобы понять действие условных инструкций, мы предлагали представить их в виде разветвлений на дороге, по которой двигается интерпретатор JavaScript. Инструкции циклов можно представить как разворот на дороге, возвращающий обратно, который заставляет интерпретатор многократно проходить через один и тот же участок программного кода.

В языке JavaScript имеется четыре инструкции циклов: **while, do/while, for и for/in.** Одно из обычных применений инструкций циклов – обход элементов массива. Эта разновидность циклов также имеет специальные методы итераций класса Array.

**Инструкция while.** Инструкция if является базовой условной инструкцией в языке JavaScript, а базовой инструкцией циклов для JavaScript можно считать инструкцию while. Она имеет следующий синтаксис:

```
while (выражение) {
      инструкция
}
```

Инструкция while **начинает работу с вычисления выражения**. Если это выражение имеет ложное значение, интерпретатор пропускает инструкцию, составляющую тело цикла, и переходит к следующей инструкции в программе. Если выражение имеет истинное значение, то выполняется инструкция, образующая тело цикла, затем управление передается в начало цикла и выражение вычисляется снова. Иными словами, интерпретатор снова и снова выполняет инструкцию тела цикла, пока (while) значение выражения остается истинным. Обратите внимание, что имеется возможность организовать **бесконечный цикл** с помощью синтаксиса while(true). Обычно не требуется, чтобы интерпретатор JavaScript снова и снова выполнял одну и ту же операцию. Почти в каждом цикле с каждой итерацией цикла одна или несколько переменных изменяют свои значения. Поскольку переменная меняется, действия, которые выполняет инструкция, при каждом проходе тела цикла могут отличаться. Кроме того, если изменяемая переменная (или переменные) присутствует в выражении, значение выражения может меняться при каждом проходе цикла. Это важно, т.к. в противном случае выражение, значение которого было истинным, никогда не изменится и цикл никогда не завершится! Ниже приводится пример цикла while, который выводит числа от 0 до 9:

```
var count = 0;
while (count < 10) {
      console.log(count);
      count++;
}
```

Как видите, в начале переменной count присваивается значение 0, а затем ее значение увеличивается каждый раз, когда выполняется тело цикла. После того как цикл будет выполнен 10 раз, выражение вернет false (т. е. переменная count уже не меньше 10), инструкция while завершится и интерпретатор переходит к следующей инструкции в программе. Большинство циклов имеют переменные-счетчики, аналогичные count. Чаще всего в качестве счетчиков цикла выступают переменные с именами i, j и k, хотя для того

чтобы сделать программный код более понятным, следует давать счетчикам более наглядные имена.

**Инструкция do/while.** Цикл do/while во многом похож на цикл while, за исключением того, что **выражение цикла проверяется в конце**, а не в начале. Это значит, что тело цикла всегда выполняется как минимум один раз. Эта инструкция имеет следующий синтаксис:

```
do {
      инструкция
} while (выражение);
```

Цикл do/while используется реже, чем родственный ему цикл while. Дело в том, что на практике ситуация, когда вы заранее уверены, что потребуется хотя бы один раз выполнится тело цикла, несколько необычна. Ниже приводится пример использования цикла do/while:

```
function printArray(a) {
      var len = a.length, i = 0;
      if (len == 0) {
            console.log("Пустой массив");
      } else {
            do {
                  console.log(a[i]);
            } while (++i < len);
      }
}
```

Между циклом do/while и обычным циклом while имеется два отличия. Во-первых, цикл do требует как ключевого слова do (для отметки начала цикла), так и ключевого слова while (для отметки конца цикла и указания условия). Во-вторых, в отличие от цикла while, цикл do завершается точкой с запятой. Цикл while не обязательно завершать точкой с запятой, если тело цикла заключено в фигурные скобки.

**Инструкция for.** Инструкция for представляет собой конструкцию цикла, которая часто оказывается более удобной, чем инструкция while. Инструкция for упрощает конструирование циклов, следующих шаблону, общему для большинства циклов. Большинство циклов имеют некоторую переменную-счетчик. Эта переменная инициализируется перед началом цикла и проверяется перед каждой итерацией. Наконец, переменная-счетчик инкрементируется или изменяется каким-либо другим образом в конце тела цикла, непосредственно перед повторной проверкой переменной. **Инициализация, проверка и обновление** – это три ключевых операции, выполняемых с переменной цикла. Инструкция for делает эти три шага явной частью синтаксиса цикла:

```
for(инициализация; проверка; инкремент) {
      инструкция
}
```

**Инициализация, проверка и инкремент** – это три выражения (разделенных точкой с запятой), которые ответственны за инициализацию, проверку и увеличение переменной цикла.

Расположение их в первой строке цикла упрощает понимание того, что делает цикл for, и не позволяет забыть инициализировать или увеличить переменную цикла. Проще всего объяснить работу цикла for, показав эквивалентный ему цикл while:

```
инициализация;
while(проверка) {
      инструкция
      инкремент;
}
```

Другими словами, выражение инициализации вычисляется один раз перед началом цикла. Это выражение, как правило, является выражением с побочными эффектами (обычно присваиванием). В JavaScript также допускается, чтобы выражение инициализации было инструкцией объявления переменной var, поэтому можно одновременно объявить и инициализировать счетчик цикла. **Выражение проверки вычисляется перед каждой итерацией** и определяет, будет ли выполняться тело цикла. Если результатом проверки является истинное значение, выполняется инструкция, являющаяся телом цикла. **В конце цикла вычисляется выражение инкремент**. Чтобы использование этого выражения имело смысл, оно должно быть выражением с побочными эффектами. Обычно это либо выражение присваивания, либо выражение, использующее оператор ++ или --. Вывести числа от 0 до 9 можно также с помощью цикла for, как показано ниже. В противовес эквивалентному циклу while, показанному в предыдущем разделе:

```
for(var count = 0; count < 10; count++) {
      console.log(count);
}
```

Конечно, циклы могут быть значительно более сложными, чем в этих простых примерах, и иногда в каждой итерации цикла изменяется несколько переменных. Эта ситуация – единственный случай в JavaScript, когда часто применяется оператор «запятая» – он позволяет объединить несколько выражений инициализации и инкрементирования в одно выражение, подходящее для использования в цикле for:

```
var i, j
for(i = 0, j = 10; i < 10; i++, j--) {
      sum += i * j;
}
```

Во всех наших примерах циклов, представленных до сих пор, переменная цикла содержала число. Это достаточно распространенная, но не обязательная практика. В следующем примере цикл for используется для обхода связанного списка структур данных и получения последнего объекта в списке (например, первого объекта, который не имеет свойства next):

```
function tail(o) { // Возвращает последний элемент в списке o
    for(; o.next; o = o.next) /* пустое */ ; // Выполнять обход, пока o.next
```

```
        return o; // является истинным значением
}
```

Обратите внимание на отсутствие выражения инициализации в примере выше. Любое из трех выражений цикла for может быть опущено, но две точки с запятой являются обязательными. Если опустить выражение проверки, цикл будет повторяться вечно, и форма записи for(;;) является еще одним способом написать бесконечный цикл, подобно while(true).

**2. Переходы**

Еще одной категорией инструкций языка JavaScript являются инструкции перехода. Как следует из названия, эти инструкции заставляют интерпретатор JavaScript переходить в другое место в программном коде. **Инструкция break** заставляет интерпретатор перейти в конец цикла или другой инструкции. **Инструкция continue** заставляет интерпретатор пропустить оставшуюся часть тела цикла, перейти обратно в начало цикла и приступить к выполнению новой итерации. В языке JavaScript имеется **возможность помечать инструкции именами**, благодаря чему в инструкциях break и continue можно явно указывать, к какому циклу или к какой другой инструкции они относятся.

**Метки инструкций.** Любая инструкция может быть помечена указанным перед ней идентификатором и двоеточием:

**идентификатор: инструкция**

Помечая инструкцию, вы тем самым даете ей имя, которое затем можно будет использовать в качестве ссылки в любом месте в программе. Пометить можно любую инструкцию, однако помечать имеет смысл только инструкции, имеющие тело, такие как циклы и условные инструкции. Присвоив имя циклу, его затем можно использовать в инструкциях break и continue, внутри цикла для выхода из него или для перехода в начало цикла, к следующей итерации. **Инструкции break и continue являются единственными инструкциями в языке JavaScript, в которых можно указывать метки**. Ниже приводится пример инструкции while с меткой и инструкции continue, использующей эту метку:

```
    mainloop: while(token != null) {
    // Программный код опущен...
    continue mainloop; // Переход к следующей итерации именованного цикла
    // Программный код опущен...
    }
```

Идентификатор, используемый в качестве метки инструкции, может быть любым допустимым идентификатором JavaScript, кроме зарезервированного слова. **Имена меток отделены от имен переменных и функций, поэтому в качестве меток допускается использовать идентификаторы, совпадающие с именами переменных или функций**. Метки инструкций определены только внутри инструкций, к которым они применяются (и, конечно же, внутри вложенных в них инструкций). Вложенные инструкции не могут помечаться теми же идентификаторами, что и вмещающие их инструкции, но две независимые инструкции могут помечаться одинаковыми метками.

Помеченные инструкции могут помечаться повторно. Тоесть любая инструкция может иметь множество меток.

**Инструкция break.** Инструкция break приводит к немедленному выходу из самого внутреннего цикла или инструкции switch. Синтаксис ее прост:

```
break;
```

Поскольку инструкция break приводит к выходу из цикла или инструкции switch, такая форма break допустима только внутри этих инструкций. Выше мы уже видели примеры использования инструкции break внутри инструкции switch. В циклах она обычно используется для немедленного выхода из цикла, когда по каким-либо причинам требуется завершить выполнение цикла. Когда цикл имеет очень сложное условие завершения, зачастую проще бывает реализовать эти условия с помощью инструкций break, чем пытаться выразить их в одном условном выражении цикла. Следующий пример пытается отыскать элемент массива с определенным значением. Цикл завершается обычным образом по достижении конца массива или с помощью инструкции break, как только будет найдено искомое значение:

```
for(var i = 0; i < a.length; i++) {
      if (a[i] == target) break;
}
```

В языке JavaScript допускается указывать имя метки за ключевым словом break (идентификатор без двоеточия):

```
break имя_метки;
```

Когда инструкция break используется с меткой, она выполняет переход в конец именованной инструкции или прекращение ее выполнения. В случае отсутствия инструкции с указанной меткой попытка использовать такую форму инструкции break **порождает синтаксическую ошибку**. Именованная инструкция не обязана быть циклом или инструкцией switch: инструкция break с меткой может выполнять «выход» из любой вмещающей ее инструкции. Объемлющая инструкция может даже быть простым блоком инструкций, заключенным в фигурные скобки исключительно с целью пометить его. Между ключевым словом break и именем метки не допускается вставлять символ перевода строки. Дело в том, что интерпретатор JavaScript автоматически вставляет пропущенные точки с запятой: если разбить строку программного кода между ключевым словом break и следующей за ним меткой, интерпретатор предположит, что имелась в виду простая форма этой инструкции без метки, и добавит точку с запятой. Инструкция break с меткой необходима, только когда требуется прервать выполнение инструкции, не являющейся ближайшим объемлющим циклом или инструкцией switch. Следующий фрагмент демонстрирует это:

```
      var matrix = getData(); // Получить 2-мерный массив
чисел откуда-нибудь
      // Найти сумму всех чисел в матрице.
      var sum = 0, success = false;
      // Пометить инструкцию, выполнение которой
требуется прервать в случае ошибки
      compute_sum: if (matrix) {
            for(var x = 0; x < matrix.length; x++) {
```

```
            var row = matrix[x];
            if (!row) break compute_sum;

            for(var y = 0; y < row.length; y++) {
                var cell = row[y];
                if (isNaN(cell)) break compute_sum;
                sum += cell;
            }

        }
        success = true;
    }
    // Здесь инструкция break выполняет переход.
    Если будет выполнено условие
    // success == false, значит, что-то не так в
    полученной матрице.
    // В противном случае переменная sum будет
    содержать сумму всех элементов матрицы.
```

Наконец, обратите внимание, что инструкция break, с меткой или без нее, не может передавать управление через границы функций. Например, нельзя пометить инструкцию объявления функции и затем использовать эту метку внутри функции.

**Инструкция continue.** Инструкция continue схожа с инструкцией break. Однако вместо выхода из цикла инструкция continue запускает новую итерацию цикла. Синтаксис инструкции continue столь же прост, как и синтаксис инструкции break:

```
continue;
```

Инструкция continue может также использоваться с меткой:

```
continue имя_метки;
```

Инструкция continue, как в форме без метки, так и с меткой, может использоваться только в теле цикла. Использование ее в любых других местах приводит к синтаксической ошибке. Когда выполняется инструкция continue, текущая итерация цикла прерывается и начинается следующая.

**Для разных типов циклов это означает разное:**

• В цикле while указанное в начале цикла выражение проверяется снова, и если оно равно true, тело цикла выполняется с начала.

• В цикле do/while происходит переход в конец цикла, где перед повторным выполнением цикла снова проверяется условие.

• В цикле for вычисляется выражение инкремента и снова вычисляется выражение проверки, чтобы определить, следует ли выполнять следующую итерацию.

Обратите внимание на различия в поведении инструкции continue в циклах while и for: цикл while возвращается непосредственно к своему условию, а цикл for сначала вычисляет выражение инкремента, а затем возвращается к условию. Ранее при обсуждении цикла for объяснялось поведение цикла for в терминах «эквивалентного»

цикла while. Поскольку инструкция continue ведет себя в этих двух циклах по-разному, точно имитировать цикл for с помощью одного цикла while невозможно.

В следующем примере показано использование инструкции continue без метки для выхода из текущей итерации цикла в случае ошибки:

```
for(i = 0; i < data.length; i++) {
      if (!data[i]) continue; // Не обрабатывать неопределенные данные
      total += data[i];
}
```

Инструкция continue, как и break, может применяться во вложенных циклах в форме, включающей метку, и тогда заново запускаемым циклом не обязательно будет цикл, непосредственно содержащий инструкцию continue. Кроме того, как и для инструкции break, **переводы строк между ключевым словом continue и именем метки не допускаются.**

**Вложенные циклы и использование меток break/continue.** Бывает нужно выйти одновременно из нескольких уровней цикла. Например, внутри цикла по i находится цикл по j, и при выполнении некоторого условия мы бы хотели выйти из обоих циклов сразу:

```
outer: for (var i = 0; i < 3; i++) {
      for (var j = 0; j < 3; j++) {
            var input = prompt('Значение в координатах '+i+','+j, '');
            // если отмена ввода или пустая строка -
            // завершить оба цикла
            if (!input) break outer; // (*)
       }
}
alert('Готово!');
```

***Чет & нечет.*** *Написать программу, выводящую на экран только четные целые числа из диапазона от 1 до 20. Тоже самое для нечетных цифр.*

***Кратность 3-м.*** *Написать программу, выводящую на экран целые числа от 1 до 20 за исключением чисел, кратных 3.*

***Сумма диапазона.*** Пользователь задает диапазон чисел. Если данные некорректны - программа повторно запрашивает ввод. Вычислить сумму чисел в заданном диапазоне.

***Разбивка числа.*** Пользователь задает число. Если данные некорректны - программа повторно запрашивает ввод. Разбить введенное число на отдельные цифры и вывести в прямом и обратном порядке.

***Переворот числа.*** Пользователь вводит число. Если данные некорректны - запрашивать данные, пока не введет правильные. Перевернуть число «физически» (например, число 2356 станет числом 6532) и вывести на экран.

***Квадрат.*** Написать программу, которая выводит на экран квадрат (пользователь вводит длину стороны), внутри таблицы 20x20, квадрат состоит из закрашенных ячеек таблицы(правилом background). Создать такую же программу, но фигура пустая.

** Пользователь задает размер таблицы и квадрата. Предусмотреть некорректный ввод данных.*

***Треугольник.*** Написать программу, которая выводит на экран равнобедренный треугольник (пользователь вводит длину стороны), внутри таблицы 20x20, треугольник состоит из закрашенных ячеек таблицы(правилом background). Создать такую же программу, но фигура пустая.

** Пользователь задает размер таблицы и треугольника. Предусмотреть некорректный ввод данных.*

***Ромб.*** *Написать программу, которая выводит на экран Написать программу, которая выводят на экран ромб (пользователь вводит (пользователь вводит диагональ), внутри таблицы 20x20, ромб состоит из закрашенных ячеек таблицы(правилом background). Создать такую же программу, но фигура пустая.*

** Пользователь задает размер таблицы и ромба. Предусмотреть некорректный ввод данных.*

***Параллелограмм.*** Написать программу, которая выводит на экран параллелограмм (пользователь размеры фигуры), внутри таблицы 20x20,параллелограмм состоит из закрашенных ячеек таблицы(правилом background). Создать такую же программу, но фигура пустая.

** Пользователь задает размер таблицы и параллелограмма. Предусмотреть некорректный ввод данных.*

***Простые числа.*** Пользователь вводит диапазон. Программа, которая выводит на экран все простые числа из диапазона. Предусмотреть некорректный ввод данных.

***Факториал.*** *Написать программу, вычисляющую факториал числа. Число вводит пользователь, предусмотреть некорректный ввод данных.*

***Степень.*** *Написать программу, вычисляющую степень числа, введенного пользователем. Предусмотреть некорректный ввод данных.*

***Циклический сдвиг.*** Осуществить циклический сдвиг введенного числа вправо на N разрядов (например, при сдвиге числа 1234 на 3 разряда получится число 2341).

***Шахматная доска.*** Вывести шахматную доску 8 * 8, с размером клеток N, белые клетки - белый фон, черные - черный. Пользователь задает размер клетки. Например, если пользователь ввел  2, то размер одной клетки доски будет - 4 ячейки таблицы. Таблица строится циклом или циклами.