Laços e iterações

Laços oferecem um jeito fácil e rápido de executar uma ação repetidas vezes. Este capítulo do guia do JavaScript abordará diferentes formas de iterações existentes no JavaScript.

Você pode pensar em um laço de repetição como um jogo onde você manda o seu personagem andar X passos em uma direção e Y passos em outra; por exemplo, a ideia "vá 5 passos para leste" pode ser expressa em um laço desta forma:

```
var passo;
for (passo = 0; passo < 5; passo++) {
     // Executa 5 vezes, com os valores de passos de 0 a 4.
     console.log('Ande um passo para o leste');
}
```

Existem várias formas diferentes de laços, mas eles essencialmente fazem a mesma coisa: repetir uma ação múltiplas vezes (inclusive você poderá repetir 0 vezes). Os vários mecanismos diferentes de laços oferecem diferentes formas de determinar quando este irá começar ou terminar. Há várias situações em que é mais fácil resolver um problema utilizando um determinado tipo de laço do que outros.

Os possíveis laços de repetição em JavaScript:

- for_statement
- do...while_statement
- while_statement
- label_statement
- break_statement
- continue_statement
- for...in_statement
- for...of_statement

Declaração for

Um laço for é repetido até que a condição especificada seja falsa. O laço for no JavaScript é similar ao Java e C. Uma declaração for é feita da seguinte maneira:

```
for ([expressaoInicial]; [condicao]; [incremento])
  declaracao
```

Quando um for é executado, ocorre o seguinte:

1. A expressão `expressao Inicial` é inicializada e, caso possível, é executada. Normalmente essa expressão inicializa um ou mais contadores, mas a sintaxe permite expressões de qualquer grau de complexidade. Podendo conter também declaração de variáveis.
2. A expressão `condicao` é avaliada. caso o resultado de `condicao` seja verdadeiro, o laço é executado. Se o valor de `condicao` é falso, então o laço terminará. Se a expressão `condicao` é omitida, a `condicao` é assumida como verdadeira.
3. A instrução é executada. Para executar múltiplas declarações, use uma declaração em bloco ({ ... }) para agrupá-las.
4. A atualização da expressão `incremento`, se houver, executa, e retorna o controle para o passo 2.

Exemplo

A função a seguir contém uma declaração `for` que contará o número de opções selecionadas em uma lista (um elemento `<select>` permite várias seleções). Dentro do `for` é declarado uma váriavel `i` inicializada com zero. A declaração `for` verifica se `i` é menor que o número de opções no elemento `<select>`, executa sucessivas declaração `if`, e incrementa `i` de um em um a cada passagem pelo laço.

```
<form name="selectForm">
  <p>
    <label for="tipoMusica">Escolha alguns tipos de música, em seguida, clique no botão abaixo:</label>
    <select id="tipoMusica" name="tipoMusica" multiple="multiple">
      <option selected="selected">R&B</option>
      <option>Jazz</option>
      <option>Blues</option>
      <option>New Age</option>
      <option>Classico</option>
      <option>Ópera</option>
    </select>
```

```
  </p>
  <p><input id="btn" type="button" value="Quantos foram selecionados?" /></p>
</form>

<script>
function howMany(selectObject) {
  var numeroSelecionadas = 0;
  for (var i = 0; i < selectObject.options.length; i++) {
    if (selectObject.options[i].selected) {
      numeroSelecionadas++;
    }
  }
  return numeroSelecionadas;
}

var btn = document.getElementById("btn");
btn.addEventListener("click", function(){
  alert('Total de opções selecionadas: ' + howMany(document.selectForm.tipoMusica))
});
</script>
```

Declaração do...while

A instrução do...while repetirá até que a condição especificada seja falsa.

```
do
  declaracao
while (condicao);
```

A instrução será executada uma vez antes da condição ser verificada. Para executar multiplas instruções utilize uma declaração de bloco ({ ... }) para agrupá-las. Caso a `condicao` seja verdadeira, então o laço será executado novamente. Ao final de cada execução, a `condicao` é verificada. Quando a condição contida no while for falsa a execução do laço é terminada e o controle é passado para a instrução seguinte a `do...while`.

Exemplo

No exemplo a seguir, o laço é executado pelo menos uma vez e irá executar até que `i` seja menor que 5.

```
do {
    i += 1;
    console.log(i);
} while (i < 5);
```

Declaração while

Uma declaração while executa suas instruções, desde que uma condição especificada seja avaliada como verdadeira. Segue uma declaração `while`:

```
while (condicao)
    declaracao
```

Se a condição se tornar falsa, a declaração dentro do laço para a execução e o controle é passado para a instrução após o laço.

O teste da condição ocorre antes que o laço seja executado. Desta forma se a condição for verdadeira o laço executará e testará a condição novamente. Se a condição for falsa o laço termina e passa o controle para as instruções após o laço.

Para executar múltiplas declarações, use uma declaração em bloco ({ ... }) para agrupar essas declarações.

Exemplo 1

O `while` a seguir executará enquanto `n` for menor que três:

```
n = 0;
x = 0;
while (n < 3) {
    n++;
    x += n;
}
```

A cada iteração, o laço incrementa `n` e adiciona este valor para `x`. Portanto, `x` e `n` recebem os seguintes valores:

- Depois de executar pela primeira vez: `n` = 1 e `x` = 1
- Depois da segunda vez: `n` = 2 e `x` = 3
- Depois da terceira vez: `n` = 3 e `x` = 6

Depois de executar pela terceira vez, a condição `n < 3` não será mais verdadeira, então o laço encerrará.

Exemplo 2

Evite laços infinitos. Tenha certeza que a condição do laço eventualmente será falsa; caso contrário, o laço nunca terminará. O while a seguir executará para sempre pois sua condição nunca será falsa:

```
while (true) {
    console.log("Olá, mundo");
}
```

Declaração label

Um label provê um identificador que permite que este seja referenciado em outro lugar no seu programa. Por exemplo, você pode usar uma label para identificar um laço, e então usar `break` ou `continue` para indicar quando o programa deverá interromper o laço ou continuar sua execução.

Segue a sintaxe da instrução label:

```
label : declaracao
```

Um label pode usar qualquer idenficador que não seja uma palavra reservada do JavaScript. Você pode identificar qualquer instrução com um label.

Exemplo

Neste exemplo, o label `markLoop` idenfica um laço `while`.

```
markLoop:
while (theMark == true) {
    facaAlgo();
}
```

Declaração break

Use break para terminar laços, `switch`, ou um conjunto que utiliza label.

- Quando você utiliza `break` sem um label, ele encerrará imediatamente o laço mais interno `while`, `do-while`, `for`, ou `switch` e transferirá o controle para a próxima instrução.
- Quando você utiliza `break` com um label, ele encerrará o label específico.

Segue a sintaxe do break:

1. `break;`
2. `break label;`

Na primeira opção será encerrado o laço de repetição mais interno ou `switch`. Já na segunda opção será encerrada o bloco de código referente a label.

Exemplo 1

O exemplo a seguir percorre os elementos de um array até que ele encontre o índice do elemento que possui o valor contido em `theValue`:

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

Exemplo 2: Utilizando break em label

```
var x = 0;
var z = 0
labelCancelaLaco: while (true) {
    console.log ("Laço exterior: " + x);
    x += 1;
```

```
    z = 1;
    while (true) {
        console.log ("Laço interior: " + z);
        z += 1;
        if (z === 10 && x === 10) {
            break labelCancelaLaco;
        } else if (z === 10) {
            break;
        }
    }
}
```

Declaração continue

A declaração continue pode ser usada para reiniciar uma instrução `while`, `do-while`, `for`, ou `label`.

- Quando você utiliza `continue` sem uma label, ele encerrará a iteração atual mais interna de uma instrução `while`, `do-while`, ou `for` e continuará a execução do laço a partir da próxima iteração. Ao contrário da instrução `break`, `continue` não encerra a execução completa do laço. Em um laço `while`, ele voltará para a condição. Em um laço `for`, ele pulará para a expressão de incrementação.
- Quando você utiliza `continue` com uma label, o `continue` será aplicado ao laço identificado por esta label.

Segue a sintaxe do `continue`:

1. `continue;`
2. `continue label;`

Exemplo 1

O exemplo a seguir mostra um laço `while` utlizando `continue` que executará quando o valor de `i` for igual a 3. Desta forma, `n` recebe os valores um, três, sete, e doze.

```
i = 0;
n = 0;
while (i < 5) {
    i++;
    if (i == 3) {
        continue;
    }
    n += i;
}
```

Exemplo 2

Uma instrução label `checkiandj` contém uma instrução label c`heckj`. Se o `continue` for executado, o programa terminará a iteração atual de `checkj` e começará a próxima iteração. Toda vez que o `continue` for executado, `checkj` recomeçará até que a condição do `while` for falsa. Quando isto ocorrer `checkiandj` executará até que sua condição seja falsa.

Se o `continue` estivesse referenciando `checkiandj`, o programa deveria continuar do topo de `checkiandj`.

```
checkiandj:
    while (i < 4) {
        console.log (i);
        i += 1;
        checkj:
            while (j > 4) {
            console.log(j);
            j -= 1;
            if ((j % 2) == 0) {
                continue checkj;
            }
            console.log (j + " é estranho.");
          }
          console.log ("i = " + i);
          console.log ("j = " + j);
      }
```

Declaração for...in

A declaração for...in executa iterações a partir de uma variável específica, percorrendo todas as propriedades de um objeto. Para cada propriedade distinta, o JavaScript executará uma iteração. Segue a sintaxe:

```
for (variavel in objeto) {
    declaracoes
}
```

Exemplo

A função a seguir recebe em seu argumento um objeto e o nome deste objeto. Então executará uma iteração para cada elemento e retornará uma lista de string, que irá conter o nome da propriedade e seu valor.

```
function dump_props(obj, obj_name) {
    var result = "";
    for (var i in obj) {
        result += obj_name + "." + i + " = " + obj[i] + "<br>";
    }
    result += "<hr>";
    return result;
}
```

Para um objeto chamado `car` com propriedades `make` e `model`, o resultado será:

```
car.make = Ford
car.model = Mustang
```

Arrays

Embora seja tentador usar esta forma para interagir com os elementos de um Array, a declaração for...in irá retornar o nome pré-definido da propriedade ao invés do seu index numérico. Assim é melhor usar o tradicional for com index numérico quando interagir com arrays, pois o for...in interage com as propriedades definidas pelo programador ao invés dos elementos do array.

Declaração for...of

A declaração for...of cria uma laço com objetos interativos ((incluindo, `Array`, `Map`, `Set`, assim por conseguinte), executando uma iteração para o valor de cada propriedade distinta.

```
for (variavel of objeto) {
    declaracoes
}
```

O exemplo a seguir mostra a diferença entre o `for...of` e o `for...in`. Enquanto o `for...in` interage com o nome das propriedades, o `for...of` interage com o valor das propriedades.

```
let arr = [3, 5, 7];
arr.foo = "hello";

for (let i in arr) {
    console.log (i); // logs "0", "1", "2", "foo"
}

for (let i of arr) {
    console.log (i); // logs "3", "5", "7"
}
```