Tomando decisões no seu código — condicionais

Samuel Amaro

13 de abril de 2021

Em qualquer linguagem de programação, o código precisa tomar decisões e realizar ações de acordo, dependendo de diferentes entradas. Por exemplo, em um jogo, se o número de vidas do jogador é 0, então o jogo acaba. Em um aplicativo de clima, se estiver sendo observado pela manhã, ele mostra um gráfico do nascer do sol; Mostra estrelas e uma lua se for noite. Neste artigo, exploraremos como as chamadas declarações condicionais funcionam em JavaScript.

Objetivo: Entender como usar estruturas condicionais em JavaScript.

Você pode tê-lo em uma condição

Seres humanos (e outros animais) tomam decisões o tempo todo que afetam suas vidas, desde pequenas (”devo comer um biscoito ou dois?”) até grandes (”devo ficar no meu país de origem e trabalhar na fazenda do meu pai ou devo mudar para a América e estudar astrofísica?”).

As declarações condicionais nos permitem representar tomadas de decisão como estas em JavaScript, a partir da escolha que deve ser feita (por exemplo, ”um biscoito ou dois”), ao resultado obtido dessas escolhas (talvez o resultado de ”comer um biscoito”possa ser ”ainda sentido fome ”, e o resultado de ”comer dois biscoitos”pode ser ”ter se sentido cheio, mas mamãe me falou para comer todos os biscoitos”.)

Declarações if...else

De longe o tipo mais comum de declaração condicional que você usará em JavaScript — as modestas declarações `if ... else`.

Sintaxe básica if...else

Veja a sintaxe básica do `if...else` no pseudocódigo:

```
1
2  if (condicao) {
3    codigo para executar caso a condi  o seja verdadeira
4  } else {
5    sen o, executar este c digo
6  }
```

Listing 1: Sintaxe básica

Aqui nós temos:

1. A palavra reservada `if` seguida de um par de parênteses.
2. Um teste condicional, localizado dentro dos parênteses (normalmente "este valor é maior que esse", ou "este valor existe"). Esta condição pode fazer uso dos operadores de comparação que discutimos no último módulo, e podem retornar `true` ou `false`.
3. Um par de chaves, e dentro dele temos código — pode ser qualquer código que queiramos, e só vai ser executado se o teste condicional retornar `true`.
4. A palavra reservada `else`.
5. Outro par de chaves, dentro dele temos mais um pouco de código — pode ser qualquer código que queiramos, e só vai executar se o teste condicional retornar um valor diferente de `true`, neste caso `not true`, ou `false`

Este tipo de código é bem legível por seres humanos — ele diz: "**if** a condição for `true`, execute o bloco de código A, **else** execute o bloco de código B"(se a condição for verdadeira, execute o bloco de código A, **senão** execute o bloco de código B).

Você precisa saber que não é obrigado a colocar a palavra reservada `else` e e o segundo bloco de par de chaves. O código apresentado a seguir é perfeitamente válido e não produz erros:

```
1  if (condicao) {
2      codigo para executar se a condi   o for verdadeira
3  }
4
5  c digo a ser executado
```

Entretanto, você precisa ser cauteloso aqui — neste caso, repare que o segundo bloco de código não é controlado pela declaração condicional, então ele vai executar **sempre**, independente do teste condicional retornar `true` ou `false`. É claro, isto não é necessariamente uma coisa ruim, mas isso pode não ser o que você quer — com muita frequência você vai querer executar ou um bloco de código ou outro, não os dois juntos.

Por fim, você verá muitas vezes declarações `if...else` escritas sem as chaves, no seguinte estilo de escrita:

```
1
2  if (condicao) executar aqui se for verdadeira
3  else executar este outro codigo
```

Este é um código perfeitamente válido, mas não é recomendado — ele facilita que você escreva código fora do escopo do `if` e do `else`, o que seria mais difícil se você estivesse usando as chaves para delimitar os blocos de código, e usando multiplas linhas de código e identação.

Um exemplo real

Para entender bem a sintaxe, vamos considerar um exemplo real. Imagine um filhote de humanos sendo chamado a ajudar com as tarefas do Pai ou da Mãe. Os pais podem falar: "Ei querido, se você me ajudar a ir e fazer as compras, eu te dou uma grana extra para que você possa comprar aquele brinquedo que você quer."Em JavaScript, nós podemos representar isso como:

```
1
2     var comprasFeitas = false;
3
4     if(comprasFeitas === true) {
5         var granaFilhote = 10;
6     }else{
7         var granaFilhote = 5;
8     }
```

Esse código como mostrado irá sempre resultar na variável `comprasFeitas` retornando `false`, sendo um desapontamento para nossas pobres crianças. Cabe a nós fornecer um mecanismo para o pai definir a variável `comprasFeitas` como `true` se o filho fez as compras.

else if

O último exemplo nos forneceu duas opções ou resultados - mas e se quisermos mais do que dois?

Existe uma maneira de encadear escolhas/resultados extras ao seu `if...else` — usando `else if`. Cada escolha extra requer um bloco adicional para colocar entre `if() { ... } e else { ... }` — confira o seguinte exemplo mais envolvido, que pode fazer parte de um aplicativo simples de previsão do tempo:

```
1
2     <label for="weather">Select the weather type today: </label>
3     <select id="weather">
4       <option value="">--Make a choice--</option>
5       <option value="sunny">Sunny</option>
6       <option value="rainy">Rainy</option>
7       <option value="snowing">Snowing</option>
8       <option value="overcast">Overcast</option>
9     </select>
10
11    <p></p>
```

```
1
2     var select = document.querySelector('select');
3     var para = document.querySelector('p');
4
5     select.addEventListener('change', setWeather);
6
7     function setWeather() {
8       var choice = select.value;
9
10      if (choice === 'sunny') {
11        para.textContent = 'It is nice and sunny outside today.
      Wear shorts! Go to the beach, or the park, and get an ice cream
      .';
12      } else if (choice === 'rainy') {
13        para.textContent = 'Rain is falling outside; take a rain
      coat and a brolly, and don\'t stay out for too long.';
14      } else if (choice === 'snowing') {
15        para.textContent = 'The snow is coming down     it is
      freezing! Best to stay in with a cup of hot chocolate, or go
      build a snowman.';
16      } else if (choice === 'overcast') {
17        para.textContent = 'It isn\'t raining, but the sky is grey
      and gloomy; it could turn any minute, so take a rain coat just
      in case.';
18      } else {
19        para.textContent = '';
20      }
21    }
```

1. Aqui, temos um elemento HTML `<select>` que nos permite fazer escolhas de clima diferentes e um simples parágrafo.

2. No JavaScript, estamos armazenando uma referência para ambos os elementos `<select>` e `<p>`, e adicionando um listener de evento ao elemento `<select>` para que, quando o valor for alterado, a função `setTempo()` é executada.

3. Quando esta função é executada, primeiro definimos uma variável chamada `escolha` para o valor atual selecionado no elemento `<select>`. Em seguida, usamos uma instrução condicional para mostrar um texto diferente dentro do parágrafo, dependendo de qual é o valor de `escolha`. Observe como todas as condições são testadas nos blocos else `if() {...}`, com exceção do primeiro, que é testado em um bloco `if() {...}`.

4. A última escolha, dentro do bloco `else {...}`, é basicamente uma opção de "último recurso"— o código dentro dele será executado se nenhuma das condições for `true`. Nesse caso, ele serve para esvaziar o texto do parágrafo, se nada for selecionado, por exemplo, se um usuário decidir selecionar novamente a opção de espaço reservado -escolha o clima–"mostrada no início.

Uma nota sobre os operadores de comparação

Operadores de comparação são usados para testar as condições dentro de nossas declarações condicionais. Nós primeiro olhamos para operadores de comparação de volta em nosso artigo Matemática básica em JavaScript - números e operadores. Nossas escolhas são:

- `===` e `!==` - testar se um valor é idêntico ou não idêntico a outro.
- `<` e `>` - teste se um valor é menor ou maior que outro.
- `<=` e `>=` - testar se um valor é menor ou igual a, ou maior que ou igual a outro.

Queríamos fazer uma menção especial do teste de valores boolean (`true/false`) , e um padrão comum que você vai encontrar de novo e de novo. Qualquer valor que não seja `false`, `undefined`, `null`, `0`, `NaN`, ou uma string vazia (`''`) retorna `true` quando testado como uma instrução condicional, portanto, você pode simplesmente usar um nome de variável para testar se é verdadeiro , ou mesmo que existe (ou seja, não é indefinido). Por exemplo:

```
1
2   var cheese = 'Cheddar';
3
4   if (cheese) {
5       console.log('Yay! Queijo dispon vel para fazer queijo na
    torrada');
```

```
    } else {
        console.log('Sem queijo na torrada para voc  hoje.');
    }
```

E, voltando ao nosso exemplo anterior sobre a criança fazendo uma tarefa para seu pai, você poderia escrevê-lo assim:

```

    var shoppingDone = false;

    if (shoppingDone) { // don't need to explicitly specify '===
    true'
      var childsAllowance = 10;
    } else {
      var childsAllowance = 5;
    }
```

Aninhando if...else

É perfeitamente correto colocar uma declaração `if...else` dentro de outra — para aninhá-las. Por exemplo, poderíamos atualizar nosso aplicativo de previsão do tempo para mostrar mais opções dependendo de qual é a temperatura:

```

 if (escolha === 'ensolarado') {
     if (temperature < 86) {
       para.textContent = '   ' + temperatura + ' graus l  fora -
    agrad vel e ensolarado. Vamos para a praia, ou para o parque,
    e tomar um sorvete.';
       }else if (temperature >= 86) {
            para.textContent = '   ' + temperatura + ' graus l
   fora     REALMENTE QUENTE! Se voc  quiser ir l  fora,
   certifique-se de colocar um pouco de creme de sol';
       }
  }
```

Mesmo que o código trabalhe em conjunto, cada uma das instruções `if...else` funcionam completamente independente uma da outra.

Operadores Lógicos: AND, OR e NOT

Se você quiser testar várias condições sem escrever instruções aninhadas `if...else`, os operadores lógicos poderão ajudá-lo. Quando usado em condições, os dois primeiros fazem o seguinte:

- `&&` - AND; permite encadear duas ou mais expressões para que todas elas tenham que ser avaliadas individualmente como `true` assim toda a expressão retorna `true`.

- —— - OR; permite encadear duas ou mais expressões para que uma ou mais delas tenham que ser avaliadas individualmente como `true` assim toda a expressão retorna `true`.

Para fornecer um exemplo AND, o fragmento de exemplo anterior pode ser reescrito assim:

```
1
2     if (escolha === 'ensolarado' && temperature < 86) {
3         paragrafo.textContent = '   ' + temperatura + ' graus l
      fora - agrad vel e ensolarado. Vamos para a praia, ou para o
      parque, e tomar um sorvete.';
4     } else if (escolha === 'ensolarado' && temperature >= 86) {
5         paragrafo.textContent = '   ' + temperatura + ' graus l
      fora     REALMENTE QUENTE! Se voc  quiser ir l  fora,
      certifique-se de colocar um pouco de creme de sol';
6     }
```

Então, por exemplo, o primeiro bloco de código só será executado se ambas as condições `escolha === 'ensolarado'` e `temperature < 86` retornarem `true`.

Vamos ver um exemplo rápido de OR:

```
1
2     if (iceCreamVanOutside || houseStatus === 'on fire') {
3       console.log('You should leave the house quickly.');
4     } else {
5       console.log('Probably should just stay in then.');
6     }
```

O último tipo de operador lógico, NOT, expressado pelo operador `!`, pode ser usado para negar uma expressão. Vamos combiná-lo com OR no exemplo acima:

```
1
2     if (!(iceCreamVanOutside || houseStatus === 'on fire')) {
3         console.log('Probably should just stay in then.');
4     } else {
5         console.log('You should leave the house quickly.');
6     }
```

Nesse trecho, se a instrução OR retornar `true`, o operador NOT negará isso para que a expressão geral retorne `false`.

Você pode combinar tantas instruções lógicas quanto quiser, em qualquer estrutura. O exemplo a seguir executa o código interno apenas se ambos os conjuntos de instruções OR retornarem true, significando que a instrução AND global também retornará true:

```
1
```

```
    if ((x === 5 || y > 3 || z <= 10) && (loggedIn || userName === 'Steve')) {
        // run the code
    }
```

Um erro comum ao usar o operador OR lógico em instruções condicionais é tentar indicar a variável cujo valor você está verificando uma vez e, em seguida, fornecer uma lista de valores que poderia ser para retornar true, separados pelos operadores —— (OR) . Por exemplo:

```

    if (x === 5 || 7 || 10 || 20) {
        // run my code
    }
```

Nesse caso, a condição dentro de if(...) sempre será avaliada como verdadeira, já que 7 (ou qualquer outro valor diferente de zero) sempre é avaliado como verdadeiro. Esta condição está realmente dizendo "se x é igual a 5, ou 7 é verdade — o que sempre é". Isso logicamente não é o que queremos! Para fazer isso funcionar, você precisa especificar um teste completo ao lado de cada operador OR:

```

    if (x === 5 || x === 7 || x === 10 ||x === 20) {
        // run my code
    }
```

Instruções switch

As instruções `if...else` fazem o trabalho de habilitar o código condicional bem, mas elas também possuem suas desvantagens. Elas são boas principalmente para casos em que você tem algumas opções, e cada uma requer uma quantidade razoável de código para ser executado, e / ou as condições são complexas (por exemplo, vários operadores lógicos). Nos casos em que você deseja definir uma variável para uma determinada opção de valor ou imprimir uma determinada instrução dependendo de uma condição, a sintaxe pode ser um pouco incômoda, especialmente se você tiver um grande número de opções.

As instruções `switch` são suas amigas aqui — elas tomam uma única expressão/valor como uma entrada e, em seguida, examinam várias opções até encontrarem um que corresponda a esse valor, executando o código correspondente que o acompanha. Aqui está mais um pseudocódigo, para você ter uma ideia:

```

    switch (expression) {
      case choice1:
```

```
        run this code
        break;

      case choice2:
        run this code instead
        break;

      // include as many cases as you like

      default:
        actually, just run this code
    }
```

Aqui nós temos:

1. A palavra-chave `switch`, seguido por um par de parênteses.
2. Uma expressão ou valor dentro dos parênteses.
3. A palavra-chave `case`, seguido por uma escolha que a expressão/valor poderia ser, seguido por dois pontos.
4. Algum código para ser executado se a escolha corresponder à expressão.
5. Uma instrução `break`, seguido de um ponto e vírgula. Se a opção anterior corresponder à expressão / valor, o navegador interromperá a execução do bloco de código aqui e passará para qualquer código que aparecer abaixo da instrução `switch`
6. Como muitos outros casos (marcadores 3 a 5) que você quiser.
7. A palavra-chave `default`, seguido por exatamente o mesmo padrão de código de um dos casos (marcadores 3 a 5), exceto que o `default` não tem escolha após ele, e você não precisa da instrução `break`, pois não há nada para executar depois disso o bloco de qualquer maneira. Esta é a opção padrão que é executada se nenhuma das opções corresponder.

Um exemplo de switch

Vamos dar uma olhada em um exemplo real — vamos reescrever nosso aplicativo de previsão do tempo para usar uma instrução switch:

```
  <label for="weather">Select the weather type today: </label>
  <select id="weather">
    <option value="">--Make a choice--</option>
    <option value="sunny">Sunny</option>
    <option value="rainy">Rainy</option>
    <option value="snowing">Snowing</option>
    <option value="overcast">Overcast</option>
```

```
    </select>

    <p></p>
```

```

    var select = document.querySelector('select');
    var para = document.querySelector('p');

    select.addEventListener('change', setWeather);


    function setWeather() {
      var choice = select.value;

      switch (choice) {
        case 'sunny':
          para.textContent = 'It is nice and sunny outside today.
    Wear shorts! Go to the beach, or the park, and get an ice cream
    .';
          break;
        case 'rainy':
          para.textContent = 'Rain is falling outside; take a rain
    coat and a brolly, and don\'t stay out for too long.';
          break;
        case 'snowing':
          para.textContent = 'The snow is coming down     it is
    freezing! Best to stay in with a cup of hot chocolate, or go
    build a snowman.';
          break;
        case 'overcast':
          para.textContent = 'It isn\'t raining, but the sky is
    grey and gloomy; it could turn any minute, so take a rain coat
    just in case.';
          break;
        default:
          para.textContent = '';
      }
    }
```

Operador ternário

Há um bit final de sintaxe que queremos apresentar a você antes de começar a brincar com alguns exemplos. O operador ternário ou condicional é um pequeno bit de sintaxe que testa uma condição e retorna um valor / expressão se for `true`, e outro caso seja `false` — isso pode ser útil em algumas situações e pode ocupar muito menos código que um bloco `if...else` se você simplesmente tiver duas opções escolhidas entre uma condição `true`/`false` condition. O pseudocódigo é assim:

```

    ( condition ) ? //execute este codigo : //excute este codigo em
     vez de
```

Então, vamos dar uma olhada em um exemplo simples:

```
1
2    var greeting = ( isBirthday ) ? 'Happy birthday Mrs. Smith
     we hope you have a great day!' : 'Good morning Mrs. Smith.';
```

Aqui temos uma variável chamada `eAniversario` — se ela for `true`, nós damos ao nosso convidado uma mensagem de feliz aniversário; se não, damos a ela a saudação diária padrão.

Exemplo de operador ternário

Você não precisa apenas definir valores de variáveis com o operador ternário; Você também pode executar funções ou linhas de código - qualquer coisa que você gosta. O exemplo ao vivo a seguir mostra um seletor de temas simples em que o estilo do site é aplicado usando um operador ternário.

Aqui nós temos um elemento `<select>` para escolher um tema (preto ou branco), além de um simples `<h1>` (en-US) para exibir um título do site. Nós também temos uma função chamada `atualizar()`, que leva duas cores como parâmetros (entradas). A cor do plano de fundo do site é definida para a primeira cor fornecida e sua cor de texto é definida para a segunda cor fornecida.

Finalmente, nós também temos um evento listener `onchange` que serve para executar uma função que contém um operador ternário. Começa com uma condição de teste — `select.value === 'black'`. Se este retornar `true`, nós executamos a função `atualizar()` com parâmetros de preto e branco, o que significa que acabamos com a cor de fundo do preto e cor do texto de branco. Se retornar `false`, nós executamos a função `atualizar()` com parâmetros de branco e preto, o que significa que a cor do site está invertida.