Bài: Vòng lặp For trong JavaScript

Xem bài học trên website để ủng hộ Kteam: Vòng lặp For trong JavaScript

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage How Kteam nhé!

Dẫn nhập

Ở bài trước, chúng ta đã cùng nhau tìm hiểu về vòng lặp với while trong JavaScript.

Trong bài này, Kteam sẽ giới thiệu đến các bạn về một loại vòng lặp mới: **vòng lặp for**

Nội dung

Để nắm được nội dung bài này, bạn cần:

- Hiểu rõ về biến trong JavaScript, các kiểu dữ liệu cơ bản như số, chuỗi
- Biết về khái niệm vòng lặp trong JavaScript
- Biết về các câu lệnh điều kiện trong JavaScript

Những nội dung mà chúng ta sẽ cùng tìm hiểu:

- Đặt vấn đề
- Vòng lặp for

Đặt vấn đề

Như đã biết, để có thể lặp đi lặp lại một công việc nào đó, thì trong JavaScript ta có thể sử dụng vòng lặp với while.

Nhưng đôi khi, với while, một vòng lặp có thể sẽ không biết trước được số lần lặp.

Giả sử, ta tạo một trò chơi, mà khi người chơi ấn phím 0 thì trò chơi sẽ kết thúc.

Khi đó, while sẽ là một lựa chọn hợp lý.

Nhưng, giả sử ta muốn quyết định số lần lặp của một quá trình nào đó, thì sao ? Vòng lặp while vẫn cho chúng ta khả năng đó.

Javascript:

```
time = 1
// 1
times = 10
// 10
while(time < times) {
 time++;
 // statement
}
// 9
```

Nhưng đó không phải là một cách hay. Giả sử, ta quên mất việc gán **times++** thì sao ? Đó sẽ là một vòng lặp "never stop".

Để có thể lặp với số lần biết trước, JavaScript hỗ trợ một cú pháp lặp với tác dụng mạnh hơn nhiều so với while. Đó là **vòng lặp for**

Vòng lặp for

Cú pháp:

```
for(<initialization>; <condition>; <final-expression>)

<statement>
```

Trong đó:

- **<initialization>**: là một (hay một nhóm) lệnh, sẽ chỉ được thực thi một lần trước khi vòng lặp được diễn ra. Sau khi kết thúc vòng lặp, các giá trị được khởi tạo (nếu có) sẽ bị loại bỏ.
- **<condition>**: là một (hoặc một nhóm) điều kiện. Cũng như **while**, các lệnh bên trong vòng lặp chỉ được thực thi khi **<condition>** là **truthy**.
- **<final-expression>**: là một (hay một nhóm) lệnh, sẽ được thực thi sau mỗi lần hoàn thành **<statement>**
- <**statement**>: là một (một khối) lệnh, sẽ được thực thi trong mỗi lần lặp

Ví dụ:

Javascript:

```
for(let i = 0; i <= 10; i++) // in ra các số tự nhiên từ 1 đến 10.
console.log(i)
// 0
// 1
// 2
// 3
// 4
// 5
// 6
// 7
// 8
// 9
// 10
// undefined
```

Sau đây, Kteam sẽ chuyển các ví dụ (ở bài trước) từ cách dùng while sang sử dụng for để bạn có cái nhìn rõ ràng hơn

Ví dụ 1: In ra các số chẵn trong khoảng [0..10]

Javascript:

```
for(let i = 2; i <= 10; i+=2) console.log(i)
// 2
// 4
// 6
// 8
// 10
// undefined
```

Ví dụ 2: Tính tổng các số trong khoảng [0..100]

- Cách 1:

Javascript:

```
Sum = 0
// 0
for(let i = 0; i <= 100; i++) Sum += i;
// 5050
Sum
// 5050
```

Hoặc, khi đã thành thạo vòng lặp for ở một mức độ nhất định, bạn có thể

- Cách 2:

Javascript:

```
Sum = 0
// 0
for(let i = 0; i <= 100; Sum += i, i++) {}
// undefined
Sum
// 5050
```

Ví dụ 3: In ra các ký tự có trong một chuỗi:

Javascript:

```
str = 'howKteam'
// 'howKteam'
for(let i = 0; i < str.length; i++) console.log(str[i])
// h
// o
// w
// K
// t
// e
// a
// m
// undefined
```

Tại sao lại nói là for "đẳng cấp" hơn while ? Vì đơn giản, for có thể biến thể một cách linh hoạt tùy theo mong muốn của người dùng.

Trong vòng lặp for, **<initialization>**; **<condition>**; **<final-expression>** và <**statement**> là không bắt buộc.

Ví dụ dưới đây là một cách để tạo ra một vòng lặp vô tận:

Javascript:

```
for(;;)
```

Ta có thể "biến hóa" vòng lặp for theo cách của mình:

- Bỏ phần **<initialization>**:

Javascript:

```
t = 1
// 1
for(;t <= 100; t *= 2) {}
// undefined
t
// 128
```

- Việc bỏ phần **<condition>** khả năng cao là sẽ tạo ra một vòng lặp vô tận. Biện pháp cải thiện là dùng **break**

Javascript:

```
t = 1
// 1
for(;;t *= 2) {if(t > 100) break}
// undefined
t
// 128
```

Nhưng cách này chỉ khiến cho code của bạn trông dài hơn chứ chẳng mang lại bất cứ lợi ích gì, do đó đừng dùng nó.

Bên cạnh đó, với **<condition>**, ta cũng có thể ghép nhiều điều kiện lại với nhau bằng các toán tử NOT, OR và AND (toán tử logic)

Dưới đây là một ví dụ mà bạn có thể tự tìm hiểu thêm

Javascript:

```
i = 0, j = 0;
// 0
for(; i + j < 10; i--, j+=2) {}
// undefined
i
// -10
j
// 20
```

Đối với phần **<final-expression>**, ta có thể bỏ qua nó nếu như cảm thấy cần thiết.

Javascript:

```
i = 0, j = 0
// 0
for(; i + j < 10;) {i-=1; j+=2}
// 20
i
// -10
j
// 20
```

Tổng kết lại, dù là vòng lặp for hay vòng lặp while sẽ đều có những lợi thế, cũng như là những nhược điểm riêng, do đó, các bạn cần lựa chọn sử dụng cho hợp lý. Trước mắt, xuất phát điểm sẽ là thành thạo cả 2.

Dưới đây sẽ là bảng so sánh giữa vòng lặp while và vòng lặp for:

Tiêu chí	Vòng lặp for	Vòng lặp while
Cú pháp	for(<initialization>; <condition>; <final-expression>) <**statement**>	while(<condition>) <statement>
Đặc điểm	Lặp lại với **số lần biết trước**	Lặp mà không biết trước số lần lặp
Điều kiện ban đầu	Không bắt buộc phải có điều kiện ban đầu (có thể bỏ <condition>). Lúc này, nó trở thành một vòng lặp vô tận cho đến khi gặp lệnh **break**	Nếu không có điều kiện ban đầu, thì chương trình báo lỗi (không thể bỏ <condition>)
Các biến trong vòng lặp	Cho phép khởi tạo biến trước vòng lặp (các biến này chỉ được khởi tạo một lần từ đầu cho đến khi kết thúc vòng lặp)	Không hỗ trợ việc khởi tạo trước vòng lặp. Việc khởi tạo bên trong vòng lặp sẽ được thực thi lại trong mỗi lần lặp (về điểm này thì for cũng tương tự)
Trường hợp sử dụng	Lặp khi đã biết trước số lần lặp	Lặp mà không biết trước số lần lặp

Vòng lặp for có nhiều biến thể, mà Kteam sẽ hướng dẫn trong tương lai khi kiến thức của các bạn đã "vững chắc" hơn.

Kết luận

Qua bài này, các bạn đã hiểu rõ hơn về vòng lặp for.

Trong bài tiếp theo, Kteam sẽ cùng bạn ôn tập Bài tập về vòng lặp trong JavaScript.

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên "**Luyện tập – Thử thách – Không ngại khó**".