



Asynchronous và AJAX

Khóa học: Webapp Building With JavaScript

Mục tiêu



- Trình bày khái niệm đồng bộ và bất đồng bộ
- Trình bày cơ chế event loop trong Javascript Engine
- Xử lý bất đồng bộ bằng Callback, Promise, Async/Await
- Giải thích được cơ chế hoạt động của AJAX
- Trình bày được các tình huống sử dụng của AJAX
- Triển khai được AJAX sử dụng Axios
- Triển khai được AJAX với các phương thức GET, POST, PUT, DELETE



Asynchronous

Xử lý bất đồng bộ

Khái niệm Đồng bộ (Synchronous)

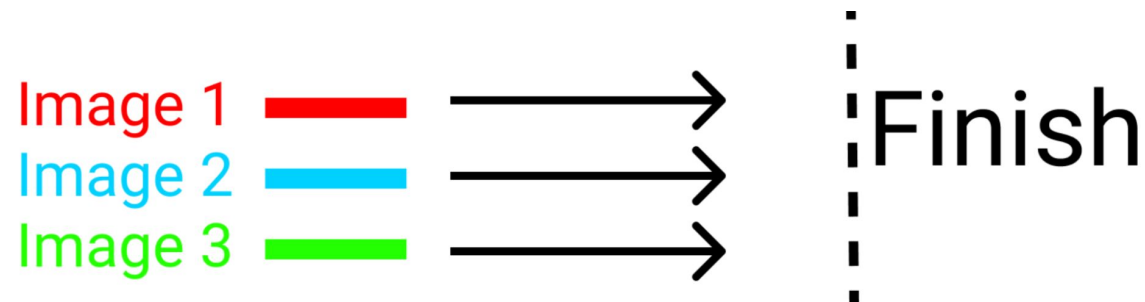
- **Synchronous (đồng bộ)** là một quy trình xử lý các công việc theo một thứ tự đã được lập sẵn. Công việc sau được bắt đầu thực hiện chỉ khi công việc thứ nhất hoàn thành.
- Trong lập trình, **xử lý đồng bộ** là mã lệnh được chạy tuần tự theo trình tự đã viết sẵn từ trên xuống dưới, mã lệnh bên dưới chỉ chạy khi mã lệnh ở bên trên đã chạy xong và trả về kết quả.



Khái niệm Bất đồng bộ (Asynchronous)



- **Asynchronous (bất đồng bộ)** là nhiều công việc có thể được thực hiện cùng lúc và nếu công việc thứ hai kết thúc trước, nó có thể sẽ cho ra kết quả trước cả câu lệnh thứ nhất..
- Trong lập trình, bất đồng bộ là các đoạn mã ở phía dưới có thể thực thi dù mã phía bên trên chưa thực thi và trả về kết quả.



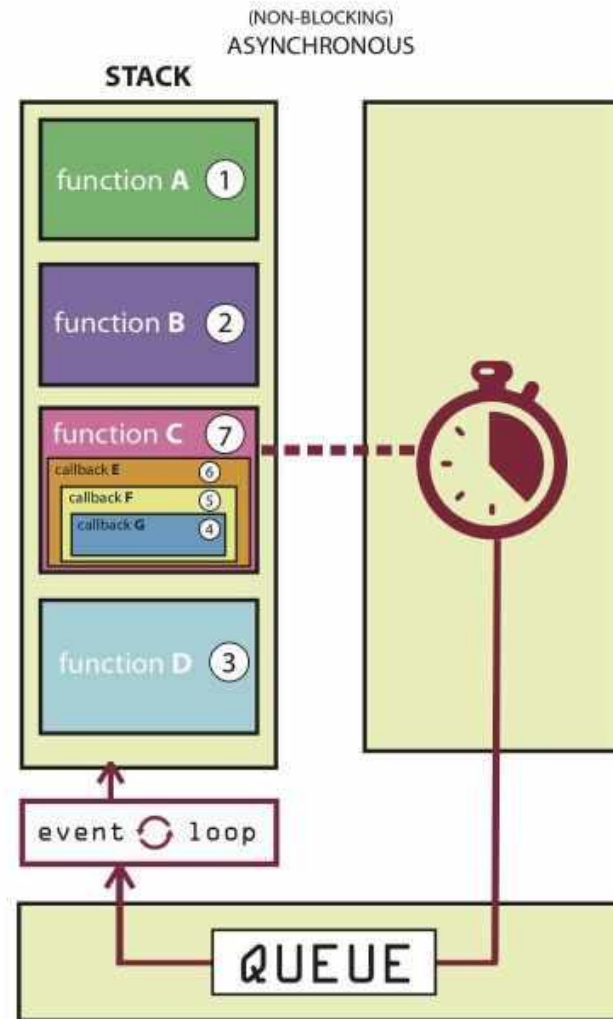
Bất đồng bộ trong JavaScript



Một câu lệnh trong JavaScript khi được thực hiện phải trải qua sự kiểm soát các các đối tượng: Timer, Message Queue, Event Loop, Call-Stack.

- Nếu một câu lệnh được gọi thao tác với WebAPI, nó sẽ được chuyển đến hàng đợi **Timer**.
- Sau khi hết thời gian chờ, nó sẽ được chuyển đến **Message Queue**.
- Khi một hàm được gọi, nó được thêm vào ngăn xếp (**Call Stack**) và sẽ được lấy ra khỏi ngăn xếp khi thực thi xong.
- **Event Loop** kiểm tra: khi nào trong **Call Stack** trống thì sẽ chuyển câu lệnh trong **Message Queue** vào **Call Stack**.

Minh họa bất đồng bộ trong JavaScript





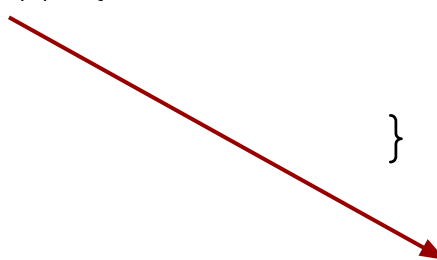
Khái niệm Callback Function

Callback Function là một hàm được gọi bởi một hàm khác.

Ví dụ:

- Hàm A được truyền vào hàm B thông qua các tham số của hàm B. Bên trong hàm B sẽ gọi đến hàm A để thực hiện một chức năng nào đó.

```
function A() {  
    // code  
}  
  
// Hàm B có một tham số callback  
function B(callback) {  
    callback();  
}  
  
B(A); //Gọi hàm B và truyền tham số là hàm A
```



Dùng callback để xử lý bất đồng bộ



Sử dụng callback là một phương pháp xử lý bất đồng bộ trong JavaScript. Khi định nghĩa hàm thực hiện một công việc tốn thời gian (ví dụ: gọi API, truy vấn cơ sở dữ liệu), chúng ta truyền thêm tham số vào hàm - đóng vai trò là hàm callback. **Khi hành động bắt đầu, kết thúc,... hàm callback sẽ được gọi ngay sau đó.**

```
function asyncFunction(callback) {  
  console.log("Start");  
  setTimeout(() => {  
    callback();  
  }, 1000);  
  console.log("Waiting");  
}
```

```
let printEnd = function() {  
  console.log("End");  
}  
  
asyncFunction(printEnd);
```




Demo

Async Callback - Dùng callback để xử lý bất đồng bộ

Hạn chế của callback - Callback Hell



```
getData("Data1", () => {  
  getData("Data2", () => {  
    getData("Data2", () => {  
      getData("Data3", () => {  
        getData("Data4", () => {  
          getData("Data5", () => {  
            getData("Data6", () => {  
              console.log("Done");  
            })  
          })  
        })  
      })  
    })  
  })  
})  
})  
})  
})
```



Khi cần thực hiện nhiều câu lệnh bất đồng bộ thì phải lồng nhiều callback với nhau, khiến cho code trở nên khó đọc, khó debug cũng như phát triển. Trường hợp này gọi là **Callback Hell**.



Promise trong JavaScript

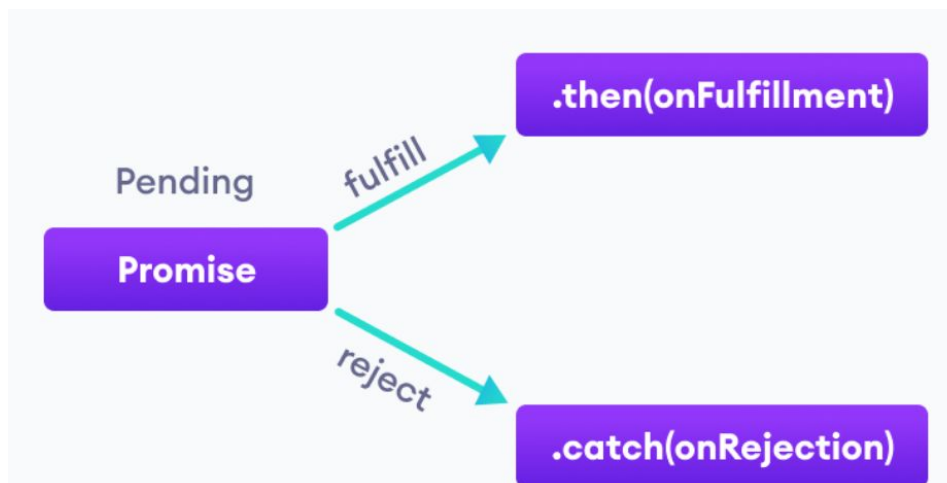
- Promise là một đối tượng bao hàm một hàm chứa các đoạn code không đồng bộ.
- Hàm này chứa 2 tham số là hai hàm callback để giải quyết sau khi mã đồng bộ thực hiện thành công hay thất bại.
- Promise cung cấp hai phương thức xử lý sau khi đoạn mã bất đồng bộ thực hiện thành công hoặc thất bại:
 - Hàm then() dùng để xử lý sau khi mã bất đồng bộ được thực hiện thành công
 - Hàm catch() dùng để xử lý sau khi mã bất đồng bộ thực hiện thất bại.

Các trạng thái của Promise

Một đối tượng Promise có thể có một trong ba trạng thái:

- **Pending**: trạng thái ban đầu, chưa rõ kết quả của thao tác là thành công hay thất bại.
- **Fulfilled**: thao tác có kết quả thành công
- **Rejected**: thao tác có kết quả thất bại

*Một promise bắt đầu ở trạng thái **Pending** - Nghĩa là quá trình này chưa hoàn tất. Nếu hoạt động thành công, quá trình kết thúc ở trạng thái **Fulfilled**. Và, nếu xảy ra lỗi, quá trình sẽ kết thúc ở trạng thái **Rejected**.*





Sử dụng Promise

Cú pháp khởi tạo Promise:

```
let promise = new Promise(function(resolve, reject) {  
    // thực hiện các công việc cụ thể  
});
```

Hàm được truyền vào new Promise gọi là *executor*. Ban đầu, Promise có trạng thái là *pending* và kết quả value là ***undefined***. Khi *executor* kết thúc công việc, nó sẽ gọi đến 1 trong 2 hàm được truyền vào:

- **resolve**(value): để xác định rằng công việc đã thực hiện thành công
 - state chuyển thành fulfilled
 - kết quả là value
- **reject**(error): để xác định rằng đã có lỗi xảy ra
 - state chuyển thành rejected
 - kết quả là error



Demo

Xử lý bất đồng bộ với Promise



Từ khóa **async/await**

Phiên bản ECMAScript 2017 đã giới thiệu hai từ khóa **async** và **await**, giúp việc viết mã xử lý bất đồng bộ trở nên dễ dàng hơn (so với cách làm trước đó là sử dụng hàm **callback** hoặc **Promise**). Giải pháp **async/await** làm cho việc sử dụng **Promise** dễ dàng hơn. Cụ thể:

- Từ khóa *async* sẽ khởi tạo một hàm trả về đối tượng **Promise**
- Từ khóa *await* sẽ khởi tạo một hàm đợi kết quả từ đối tượng **Promise**



Sử dụng từ khóa `async`

Ví dụ, chúng ta có hàm `async` sau:

```
async function myFunction() {  
    return "Hello";  
}
```

Tương đương với khai báo hàm trả về Promise như sau:

```
function myFunction() {  
    return Promise.resolve("Hello");  
}
```

Để xử lý kết quả của hàm *async*, chúng ta sử dụng phương thức `.then()` và `.catch()` trong Promise.



Sử dụng từ khóa *await*

Câu lệnh sử dụng từ khóa *await* bắt buộc nằm trong một hàm *async*.

Cú pháp: `let value = await promise;`

Ví dụ:

```
async function asyncFunc() {  
    let result1 = await promise1;  
    let result2 = await promise2;  
    let result3 = await promise3;  
  
    console.log(result1);  
    console.log(result1);  
    console.log(result1);  
}
```

Lợi ích của việc sử dụng `async/await`



- Mã dễ đọc hơn (So với sử dụng callback function hoặc Promise)
- Xử lý lỗi (error handling) đơn giản hơn
 - Chúng ta có thể sử dụng cú pháp `try..catch` có sẵn của JavaScript
- Debug dễ dàng hơn



Demo

Xử lý bất đồng bộ với `async/await`

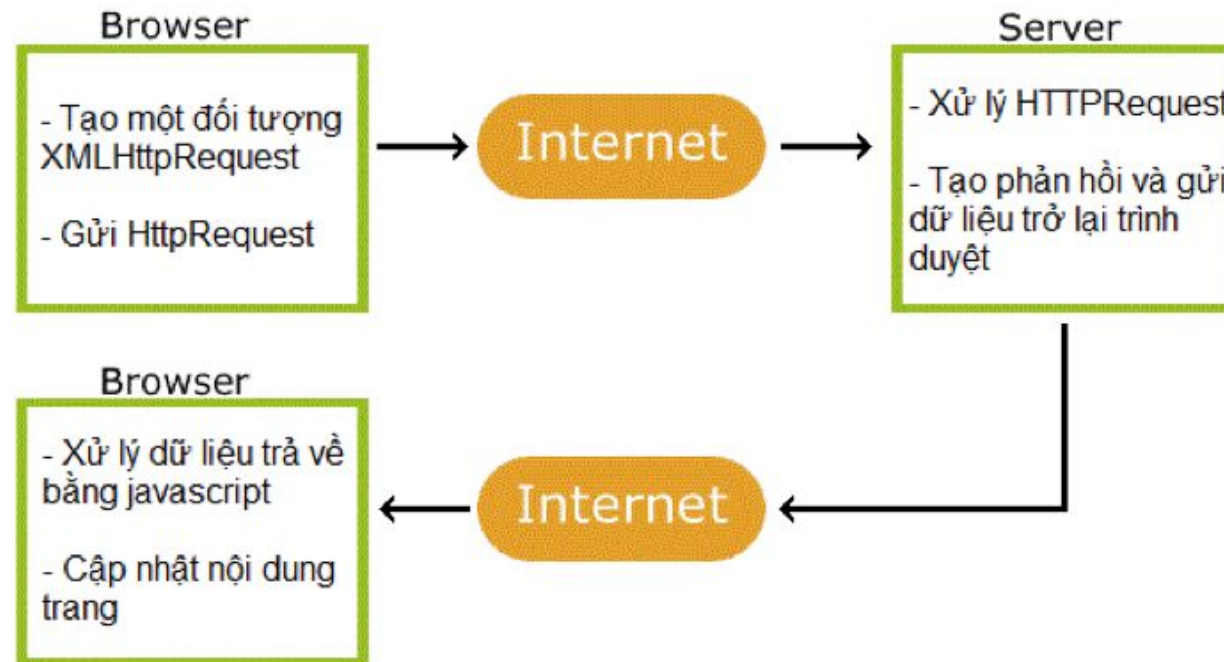


AJAX

Khái niệm AJAX



AJAX là cụm từ viết tắt của **Asynchronous JavaScript and XML**, nghĩa là JavaScript và XML không đồng bộ, là một kỹ thuật cho phép tải về nội dung của một hoặc nhiều trang mới mà không cần phải tải lại toàn bộ nội dung của trang hiện tại.





Triển khai AJAX trong dự án web

Để triển khai kỹ thuật AJAX trong các dự án web, chúng ta có những giải pháp sau:

- Sử dụng đối tượng XMLHttpRequest (có sẵn)
- Sử dụng hàm fetch (có sẵn)
- Sử dụng thư viện bên thứ ba: jQuery, Axios,...



Giới thiệu Axios

Axios là một thư viện JavaScript hỗ trợ thực hiện các yêu cầu HTTP. Nó hỗ trợ tất cả các trình duyệt hiện đại. Axios dựa trên Promise và nhờ đó, chúng ta có thể viết mã bất đồng bộ để thực hiện các yêu cầu XHR dễ dàng.

So với hàm fetch có sẵn, Axios có một số ưu điểm sau:

- Hỗ trợ một số trình duyệt cũ hơn
- Có thể hủy yêu cầu HTTP
- Có thể đặt thời gian chờ phản hồi
- Tích hợp bảo vệ CSRF
- Tự động chuyển đổi dữ liệu JSON



Demo

Sử dụng thư viện Axios

Tóm tắt



Qua bài học này, chúng ta đã tìm hiểu:

- Khái niệm đồng bộ và bất đồng bộ
- Các phương pháp xử lý bất đồng bộ trong JavaScript
 - Hàm callback
 - Promise
 - Sử dụng từ khóa `async/await`
- Khái niệm AJAX
- Triển khai AJAX trong dự án web với thư viện Axios



Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập