JavaScript Optional Chaining Operator

Summary: in this tutorial, you'll learn about the optional chaining operator (`?.`) that simplifies the way to access values through connected objects.

Introduction to the JavaScript optional chaining operator

The optional chaining operator (`?.`) is like a shortcut for accessing nested properties in a series of objects. Instead of having to check if each step in the chain is empty (`null` or `undefined`), you can use the operator `?.` to directly access the desired property.

If any part of the chain is empty, the optional chaining operator (`?.`) will stop right there and give you `undefined` as a result. It saves you from writing extra checks for each step in the chain.

Suppose that you have a function that returns a `user` object:

```
function getUser(id) {

    if(id <= 0) {
        return null;
    }

    // get the user from database
    // and return null if id does not exist
    // ...

    // if user was found, return the user
    return {
        id: id,
        username: 'admin',
        profile: {
            avatar: '/avatar.png',
            language: 'English'
        }
    }
}
```

The following uses the `getUser()` function to access the user profile:

```
let user = getUser(1);
let profile = user.profile;
```

However, if you pass the `id` that is less than or equal to zero or the `id` doesn't exist in the database, the `getUser()` function will return `null` .

Therefore, before accessing the `avatar` property, you need to check if the `user` is not `null` using the logical operator AND:

```
let user = getUser(2);
let profile = user && user.profile;
```

In this example, we confirm that the `user` is not `null` or `undefined` before accessing the value of `user.profile` property. It prevents the error that would occur if you simply access the `user.profile` directly without checking the user first.

ES2020 introduced the optional chaining operator denoted by the question mark followed by a dot:

```
?.
```

To access a property of an object using the optional chaining operator, you use one of the following:

```
objectName ?. propertyName
```

```
objectName ?. [expression]
```

The optional chaining operator implicitly checks if the `user` is not `null` or `undefined` before attempting to access the `user.profile` :

```
let user = getUser(2);
let profile = user ?. profile;
```

In this example, if the `user` is `null` or `undefined` , the optional chaining operator (`?.`) immediately returns `undefined` .

Technically, it is equivalent to the following:

```
let user = getUser(2);
let profile = (user !== null || user !== undefined)
            ? user.profile
            : undefined;
```

Stacking the optional chaining operator

In case the `user` object returned by the `getUser()` does not have the `profile` property. Trying to access the `avatar` without checking the `user.profile` first will result in an error.

To avoid the error, you can use the optional chaining operator multiple times like this:

```
let user = getUser(-1);
let avatar = user ?. profile ?. avatar;
```

In this case, the `avatar` is `undefined` .

Combining with the nullish coalescing operator

If you want to assign a default profile to the `user` , you can combine the optional chaining operator (`?.`) with the nullish coalescing operator (`??`) as follows:

```
let defaultProfile =  { default: '/default.png', language: 'English'};

let user = getUser(2);
let profile = user ?. profile ?? defaultProfile;
```

In this example, if the `user.profile` is `null` or `undefined` , the profile will take the `defaultProfile` due to the nullish coalescing operator:

Using optional chaining operator with function calls

Suppose that you have a file API as follows:

```
let file = {
    read() {
        return 'file content';
    },
    write(content) {
        console.log(`Writing ${content} to file...`);
        return true;
    }
};
```

This example calls the `read()` method of the `file` object:

```
let data = file.read();
console.log(data);
```

If you call a method that doesn't exist in the `file` object, you'll get a `TypeError` :

```
let compressedData = file.compress();
```

Error:

```
Uncaught TypeError: file.compress is not a function
```

However, if you use the optional chaining operator with the method call, the expression will return `undefined` instead of throwing an error:

```
let compressedData = file.compress?.();
```

The `compressedData` is now `undefined`.

This is useful when you use an API in which a method might be not available for some reason e.g., a specific browser or device.

The following illustrates the syntax for using the optional chaining operator with a function or method call:

```
functionName ?. (args)
```

The optional chaining operator (`?.`) is also helpful if you have a function with an optional callback:

```
function getUser(id, callback) {
    // get user
    // ...

    let user = {
        id: id,
        username: 'admin'
    };

    // test if the callback exists
    if ( callback ) {
        callback(user);
    }

    return user;
}
```

By using the optional chaining operator, you can skip the test if the callback exists:

```
function getUser(id, callback) {
    // get user
    // ...

    let user = {
        id: id,
        username: 'admin'
    };

    // test if the callback exists
    callback ?. (user);


    return user;
}
```

Summary

- The optional chaining operator (`?.`) returns `undefined` instead of throwing an error if you attempt to access a property of an `null` or `undefined` object: `obj ?. property`.
- Combine the optional chaining operator (`?.`) with the nullish coalescing operator (`??`) to assign a default value.

- Use `functionName ?. (args)` to avoid explicitly checking if the `functionName` is not `undefined` or `null` before invoking it.