



The "non-existing property" problem

Optional chaining

Short-circuiting

Other variants: ?.(), ?.[]

Summary

Comments



Edit on GitHub



4



↑ The JavaScript language → Objects: the basics

# **Optional chaining '?.'**

#### A recent addition

This is a recent addition to the language. Old browsers may need polyfills.

The optional chaining ?. is a safe way to access nested object properties, even if an intermediate properties.

## The "non-existing property" problem

If you've just started to read the tutorial and learn JavaScript, maybe the problem hasn't touched you ye

As an example, let's say we have user objects that hold the information about our users.

Most of our users have addresses in user.address property, with the street user.address.street provide them.

In such case, when we attempt to get user.address.street, and the user happens to be without an

```
1 let user = {}; // a user without "address" property
3 alert(user.address.street); // Error!
```

That's the expected result. JavaScript works like this. As user.address is undefined, an attempt to user.address.street fails with an error.

In many practical cases we'd prefer to get undefined instead of an error here (meaning "no street").

...And another example. In the web development, we can get an object that corresponds to a web page





The "non-existing property" problem

Optional chaining

Short-circuiting

Other variants: ?.(), ?.[]

Summary

Comments

Share



Edit on GitHub



3 alert(user.address ? user.address.street : undefined);



It works, there's no error... But it's quite inelegant. As you can see, the "user.address" appears twice deeply nested properties, that becomes a problem as more repetitions are required.

E.g. let's try getting user.address.street.name.

1 let user = {};

We need to check both user.address and user.address.street:

```
1 let user = {}; // user has no address
2
3 alert(user.address ? user.address.street ? user.address.street.name : r
```

That's just awful, one may even have problems understanding such code.

Don't even care to, as there's a better way to write it, using the && operator:

```
1 let user = {}; // user has no address
2
3 alert( user.address && user.address.street && user.address.street.name
```

AND'ing the whole path to the property ensures that all components exist (if not, the evaluation stops),

As you can see, property names are still duplicated in the code. E.g. in the code above, user.address

That's why the optional chaining ?. was added to the language. To solve this problem once and for all

## **Optional chaining**

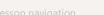
The optional chaining ?. stops the evaluation if the value before ?. is undefined or null and ret

Further in this article, for brevity, we'll be saying that something "exists" if it's not null and not

In other words, value?.prop:

· works as value nron if value evicts





The "non-existing property" problem

Optional chaining

Short-circuiting

Other variants: ?.(), ?.[]

Summary

Comments



Edit on GitHub



A

3 alert( user?.address ); // undefined

1 let user = null;

4 alert( user?.address.street ); // undefined

Please note: the ?. syntax makes optional the value before it, but not any further.

E.g. in user?.address.street.name the ?. allows user to safely be null/undefined (and retur case), but that's only for user . Further properties are accessed in a regular way. If we want some of the we'll need to replace more . with ?..



### Don't overuse the optional chaining

We should use ?. only where it's ok that something doesn't exist.

For example, if according to our coding logic user object must exist, but address is optional, th user.address?.street, but not user?.address?.street.

So, if user happens to be undefined due to a mistake, we'll see a programming error about it and coding errors can be silenced where not appropriate, and become more difficult to debug.



#### A The variable before ?. must be declared

If there's no variable user at all, then user?.anything triggers an error:

```
1 // ReferenceError: user is not defined
2 user?.address;
```

The variable must be declared (e.g. let/const/var user or as a function parameter). The option only for declared variables.

## **Short-circuiting**



Lesson navigation

The "non-existing property" problem

Optional chaining

Short-circuiting

Other variants: ?.(), ?.[]

Summary

Comments

Share



Edit on GitHub



4

## Other variants: ?.(), ?.[]

The optional chaining ?. is not an operator, but a special syntax construct, that also works with function

For example, ?.() is used to call a function that may not exist.

In the code below, some of our users have admin method, and some don't:

```
let userAdmin = {
   admin() {
      alert("I am admin");
   }
};

let userGuest = {};

userAdmin.admin?.(); // I am admin

userGuest.admin?.(); // nothing (no such method)
```

Here, in both lines we first use the dot (userAdmin.admin) to get admin property, because we assur exists, so it's safe read from it.

Then ?.() checks the left part: if the admin function exists, then it runs (that's so for userAdmin). Other evaluation stops without errors.

The <code>?.[]</code> syntax also works, if we'd like to use brackets <code>[]</code> to access properties instead of dot <code>..Sim</code> allows to safely read a property from an object that may not exist.

```
1 let key = "firstName";
2
3 let user1 = {
4    firstName: "John"
5  };
6
7 let user2 = null;
8
9 alert( user1?.[key] ); // John
10 alert( user2?.[kev] ): // undefined
```



Lesson navigation

The "non-existing property" problem

Optional chaining

Short-circuiting

Other variants: ?.(), ?.[]

Summary

Comments



Edit on GitHub



4

⚠ We can use ?. for safe reading and deleting, but not writing

The optional chaining ?. has no use at the left side of an assignment.

For example:

```
1 let user = null;
2
3 user?.name = "John"; // Error, doesn't work
4 // because it evaluates to undefined = "John"
```

It's just not that smart.

## **Summary**

The optional chaining ?. syntax has three forms:

- 1. obj?.prop returns obj.prop if obj exists, otherwise undefined.
- 2. obj?.[prop] returns obj[prop] if obj exists, otherwise undefined.
- 3. obj.method?.() calls obj.method() if obj.method exists, otherwise returns undefined.

As we can see, all of them are straightforward and simple to use. The ?. checks the left part for null/ the evaluation to proceed if it's not so.

A chain of ?. allows to safely access nested properties.

Still, we should apply  $\ \$ ?. carefully, only where it's acceptable that the left part doesn't exist. So that it v errors from us, if they occur.



reac