How to avoid 'cannot read property of undefined' errors?

Asked 6 years, 7 months ago Active 20 days ago Viewed 189k times

96

21

In my code, I deal with an array that has some entries with many objects nested inside one another, where as some do not. It looks something like the following:

```
// where this array is hundreds of entries long, with a mix
// of the two examples given
var test = [{'a':{'b':{'c':"foo"}}}, {'a': "bar"}];
```

This is giving me problems because I need to iterate through the array at times, and the inconsistency is throwing me errors like so:

```
for (i=0; i<test.length; i++) {
    // ok on i==0, but 'cannot read property of undefined' on i==1
    console.log(a.b.c);
}
```

I am aware that I can say `if(a.b){ console.log(a.b.c)}` , but this is extraordinarily tedious in cases where there are up to 5 or 6 objects nested within one another. Is there any other (easier) way that I can have it ONLY do the console.log if it exists, but without throwing an error?

javascript

edited Feb 8 '13 at 22:27
Mörre
4,801 5 31 59

asked Feb 8 '13 at 22:16
Ari
1,623 4 19 39

3 The error is probably a regular Javascript exception, so try the `try..catch` statement. That said, an array that contains wildly heterogenous elements looks like a design issue to me. – millimoose Feb 8 '13 at 22:17

2 Why would you access non-existing properties, why don't you know how the objects look like? – Bergi Feb 8 '13 at 22:22

7 I can understand why somebody would not want an error to crash everything. You can't always rely on an object's properties to exist or not exist. If you have something in place that can handle the event that the object is malformed, then you're code is much more efficient and less fragile. – SSH This Apr 9 '13 at 20:19

1 You'd be surprised at how many objects/arrays are malformed in real life situations – OneMoreQuestion Sep 8 '17 at 2:42

15 Answers

76

A quick workaround is using a try/catch helper function with ES6 arrow function:

```
function getSafe(fn, defaultVal) {
    try {
        return fn();
    } catch (e) {
        return defaultVal;
    }
}

// use it like this
getSafe(() => obj.a.lot.of.properties);

// or add an optional default value
getSafe(() => obj.a.lot.of.properties, 'nothing');
```

Working snippet:

Show code snippet

See this article for details.

There is also an ECMAScript proposal called Optional Chaining for JavaScript (currently at stage 3) which would make this even easier:

```
obj?.a?.lot?.of?.properties
```

edited Sep 11 at 9:40

answered Feb 20 '17 at 16:23

str

21.5k 9 64 93

1 I love it! the only thing I would add is a console.warn inside the catch, so that you know of the error but it continues on. – Rabbi Shuki Gur Dec 11 '18 at 10:31

Catching *all* exceptions without re-throwing is bad, and generally using exceptions as part of the expected flow of execution is also not great -- even though in this case it's pretty well contained. – hugo Aug 25 at 11:28

43

What you are doing raises an exception (and rightfully so).

You can always do

```
try{
   window.a.b.c
}catch(e){
   console.log("YO",e)
}
```

But I wouldn't, instead think of your use case.

Why are you accessing data, 6 levels nested that you are unfamiliar of? What use case justifies this?

Usually, you'd like to actually validate what sort of object you're dealing with.

Also, on a side note you should not use statements like `if(a.b)` because it will return false if a.b is 0 or even if it is "0". Instead check if `a.b !== undefined`

edited Jan 13 '17 at 13:04

rob74

3,149 16 27

answered Feb 8 '13 at 22:18

Benjamin Gruenbaum

202k 66 427 452

1 In regards to your first edit: It is justified; I am dealing with JSON structured database entries, such that the objects will scale multiple levels of fields (ie. entries.users.messages.date etc., where not all cases have data entered) – Ari Feb 8 '13 at 22:23

@Ian yeah, I obviously meant it the other way around, it will return false even if a.b is "0". Nice catch – Benjamin Gruenbaum Feb 8 '13 at 22:24

@BenjaminGruenbaum Sounds good, wasn't sure if you meant that. Also, I think you want `typeof a.b !` == "undefined" && a.b!=null` - notice the `!==` – Ian Feb 8 '13 at 22:26

3 If you don't want to tedium of a.b && a.b.c && console.log(a.b.c) then this is the only way to consistently log unknowns. – Brian Cray Feb 8 '13 at 22:31

14

If I am understanding your question correctly, you want the safest way to determine if an object contains a property.

The easiest way is using the "in" statement.

```
window.a = "aString";
//window should have 'a' property
//lets test if it exists
if ("a" in window){
    //true
 }

if ("b" in window){
     //false
 }
```

Of course you can nest this as deep as you want

```
if ("a" in window.b.c) { }
```

Not sure if this helps.

answered Feb 8 '13 at 23:16

matt weiss
327 1 4

5 You can't safely nest this as deeply as you want. What if `window.b` is undefined? You will get a type error: `Cannot use 'in' operator to search for 'c' in undefined` – Trevor Aug 28 '17 at 23:34

12

```
var testObject = {a: {b: {c: 'walrus'}}};
if(_.has(testObject, 'a.b.c')) {
  //Safely access your walrus here
}
```

edited Jun 14 '16 at 14:50

dhaupin
1,053 10 21

answered Aug 7 '15 at 19:05

tehwalris
358 4 9

2 Best, we can use `_.get()` with default for easy read : `_.get(object, 'a.b.c', 'default');` – lfnot Jun 18 '18 at 13:53

8

Try this. If `a.b` is undefined, it will leave the `if` statement without any exception.

```
if (a.b && a.b.c) {
  console.log(a.b.c);
}
```

edited Aug 25 at 23:54

customcommander
4,404 1 17 34

answered Jun 28 '17 at 11:15

Sean Chen
169 2 2

5

This is a common issue when working with deep or complex json object, so I try to avoid try/catch or embedding multiple checks which would make the code unreadable, I usually use this little piece of code in all my procect to do the job.

```
/* ex: getProperty(myObj,'aze.xyz',0) // return myObj.aze.xyz safely
 * accepts array for property names:
 *     getProperty(myObj,['aze','xyz'],{value: null})
 */
function getProperty(obj, props, defaultValue) {
    var res, isvoid = function(x){return typeof x === "undefined" || x === null;}
    if(!isvoid(obj)){
        if(isvoid(props)) props = [];
        if(typeof props  === "string") props = props.trim().split(".");
        if(props.constructor === Array){
```

```
    return typeof res === "undefined" ? defaultValue: res;
}
```

answered Sep 22 '16 at 13:53

Maelkhor

51 1 1

4

I use undefsafe religiously. It tests each level down into your object until it either gets the value you asked for, or it returns "undefined". But never errors.

answered May 12 '16 at 16:32

martinedwards

3,942 1 23 26

2 that is similar to lodash `_.get` – Filype Jul 21 '16 at 3:32

Good shout! Still useful if you don't need the other features of lodash. – martinedwards Jul 22 '16 at 14:04

3

I like Cao Shouguang's answer, but I am not fond of passing a function as parameter into the getSafe function each time I do the call. I have modified the getSafe function to accept simple parameters and pure ES5.

```
/**
* Safely get object properties.
* @param {*} prop The property of the object to retrieve
* @param {*} defaultVal The value returned if the property value does not exist
* @returns If property of object exists it is returned,
*          else the default value is returned.
* @example
* var myObj = {a : {b : 'c'} };
* var value;
*
* value = getSafe(myObj.a.b,'No Value'); //returns c
* value = getSafe(myObj.a.x,'No Value'); //returns 'No Value'
*
* if (getSafe(myObj.a.x, false)){
```

```
* }; //logs 'Not Found'
*
* if(value = getSafe(myObj.a.b, false)){
*  console.log('New Value is', value); //logs 'New Value is c'
* }
*/
function getSafe(prop, defaultVal) {
  return function(fn, defaultVal) {
    try {
      if (fn() === undefined) {
        return defaultVal;
      } else {
        return fn();
      }
    } catch (e) {
      return defaultVal;
    }
  }(function() {return prop}, defaultVal);
}
```

edited Jan 10 at 7:23

answered Jan 9 at 16:15

Hardy Le Roux

759 3 12 26

It don't really work with `getSafe(myObj.x.c)` . Tried latest versions of Chrome and Firefox. – Rickard Elimää Jan 30 at 16:26

2

In str's answer, value 'undefined' will be returned instead of the set default value if the property is undefined. This sometimes can cause bugs. The following will make sure the defaultVal will always be returned when either the property or the object is undefined.

```
const temp = {};
console.log(getSafe(()=>temp.prop, '0'));

function getSafe(fn, defaultVal) {
    try {
        if (fn() === undefined) {
            return defaultVal
        } else {
            return fn();
        }
```

```
}
```

answered Jun 28 '18 at 6:37

Cao Shouguang
21 1

2

If you use Babel, you can already use the optional chaining syntax with @babel/plugin-proposal-optional-chaining Babel plugin. This would allow you to replace this:

```
console.log(a && a.b && a.b.c);
```

with this:

```
console.log(a?.b?.c);
```

answered Feb 24 at 23:33

Fleischpflanzerl
5,013 2 28 55

1

Lodash has a `get` method which allows for a default as an optional third parameter, as show below:

```
const myObject = {
  has: 'some',
  missing: {
    vars: true
  }
}
const path = 'missing.const.value';
const myValue = _.get(myObject, path, 'default');
console.log(myValue) // prints out default, which is specified above
```

Run code snippet Expand snippet

answered Jan 30 at 13:30

Pureferret
3,544 10 56 114

1

If you have lodash you can use its `.get` method

```
_.get(a, 'b.c.d.e')
```

or give it a default value

```
_.get(a, 'b.c.d.e', default)
```

answered May 17 at 19:33

Brandon Dyer
374 1 12

1

Imagine that we want to apply a series of functions to `x` if and only if `x` is non-null:

```
if (x !== null) x = a(x);
if (x !== null) x = b(x);
if (x !== null) x = c(x);
```

Now let's say that we need to do the same to `y`:

```
if (y !== null) y = a(y);
if (y !== null) y = b(y);
if (y !== null) y = c(y);
```

```
if (z !== null) z = b(z);
if (z !== null) z = c(z);
```

As you can see without a proper abstraction, we'll end up duplicating code over and over again. Such an abstraction already exists: the *Maybe* monad.

The *Maybe* monad holds both a value and a computational context:

1. The monad keeps the value safe and applies functions to it.
2. The computational context is a null check before applying a function.

A naive implementation would look like this:

⚠ *This implementation is for illustration purpose only! This is not how it should be done and is wrong at many levels. However this should give you a better idea of what I am talking about.*

As you can see nothing can break:

1. We apply a series of functions to our value
2. If at any point, the value becomes null (or undefined) we just don't apply any function anymore.

```
const abc = obj =>
  Maybe
    .of(obj)
    .map(o => o.a)
    .map(o => o.b)
    .map(o => o.c)
    .value;

const values = [
  {},
  {a: {}},
  {a: {b: {}}},
  {a: {b: {c: 42}}}
];

console.log(

  values.map(abc)
```

```
<script>
function Maybe(x) {
  this.value = x; //-> container for our value
}

Maybe.of = x => new Maybe(x);

Maybe.prototype.map = function (fn) {
  if (this.value == null) { //-> computational context
    return this;
  }
  return Maybe.of(fn(this.value));
};
</script>
```

Run code snippet Expand snippet

Appendix 1

I cannot explain what monads are as this is not the purpose of this post and there are people out there better at this than I am. However as Eric Elliot said in hist blog post JavaScript Monads Made Simple:

> Regardless of your skill level or understanding of category theory, using monads makes your code easier to work with. Failing to take advantage of monads may make your code harder to work with (e.g., callback hell, nested conditional branches, more verbosity).

Appendix 2

Here's how I'd solve your issue using the *Maybe* monad from `monetjs`

```
const prop = key => obj => Maybe.fromNull(obj[key]);

const abc = obj =>
  Maybe
    .fromNull(obj)
    .flatMap(prop('a'))
```

```
const values = [
  {},
  {a: {}},
  {a: {b: {}}},
  {a: {b: {c: 42}}}
];

console.log(

  values.map(abc)

);


<script src="https://www.unpkg.com/monet@0.9.0/dist/monet.js"></script>
<script>const {Maybe} = Monet;</script>
```

Run code snippet Expand snippet

answered Aug 25 at 22:49

customcommander
4,404 1 17 34

0

I answered this before and happened to be doing a similar check today. A simplification to check if a nested dotted property exists. You could modify this to return the value, or some default to accomplish your goal.

```
function containsProperty(instance, propertyName) {
    // make an array of properties to walk through because propertyName can be nested
    // ex "test.test2.test.test"
    let walkArr = propertyName.indexOf('.') > 0 ? propertyName.split('.') :
[propertyName];

    // walk the tree - if any property does not exist then return false
    for (let treeDepth = 0, maxDepth = walkArr.length; treeDepth < maxDepth;
treeDepth++) {

        // property does not exist
```

```
            // does it exist - reassign the leaf
            instance = instance[walkArr[treeDepth]];


        }

        // default
        return true;

    }
```

In your question you could do something like:

```
let test = [{'a':{'b':{'c':"foo"}}}, {'a': "bar"}];
containsProperty(test[0], 'a.b.c');
```

answered Jul 20 '18 at 21:12

matt weiss
327 1 4

-1

I usually use like this:

```
var x = object.any ? object.any.a : 'def';
```

answered Dec 29 '18 at 23:07

Vansuita Jr.
1,030 10 14
