

Top 7 JavaScript Object Destructuring Techniques



Chameera Dulanga



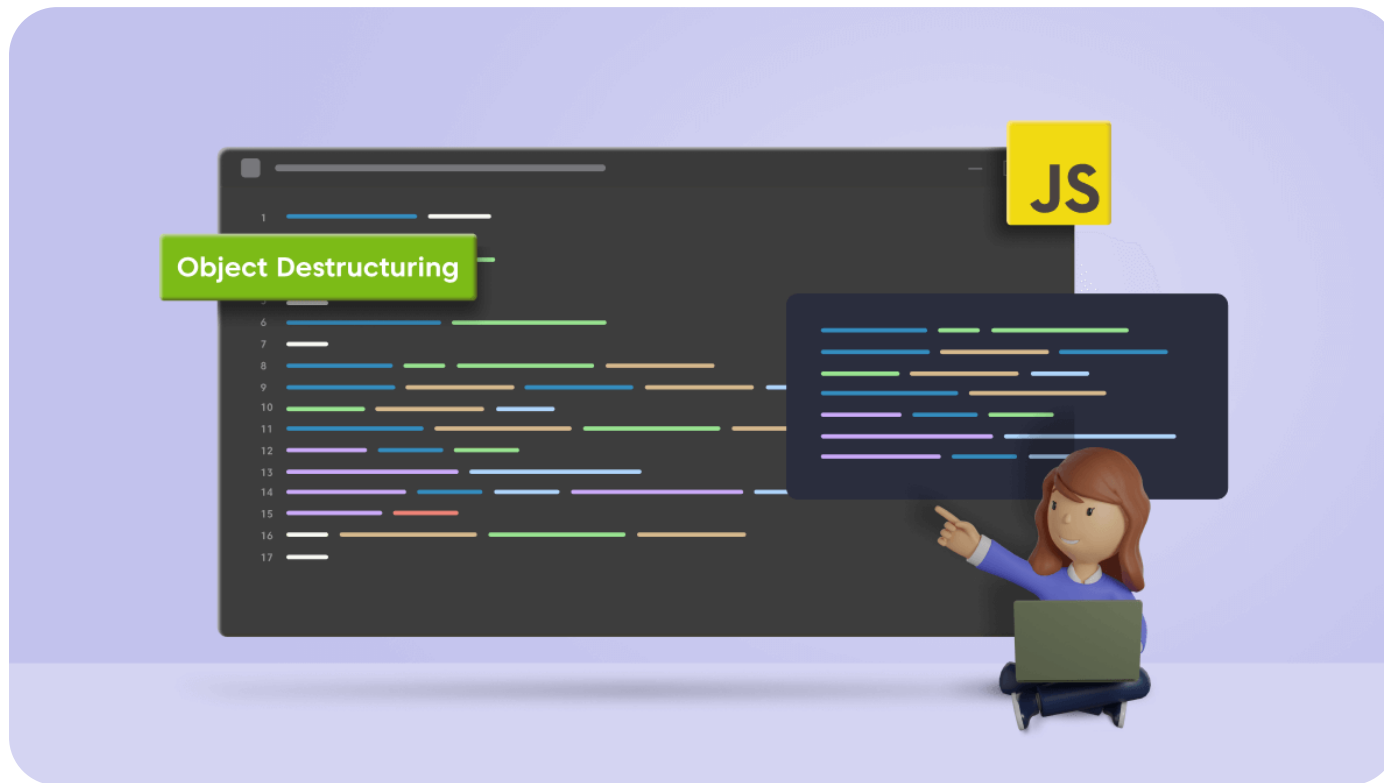
6 min read



Nov 19, 2024

Updated

1 Comments



JavaScript object destructuring was introduced with the ES6 update to help developers extract multiple properties from objects at once. This was revolutionary since objects are widely used in JavaScript as a complex data structure.

There are multiple object destructuring techniques available in JavaScript. As developers, we must understand these techniques to write clean code and improve our efficiency.

This article will discuss the top seven JavaScript object destructuring techniques to give you a better understanding of how and when they should be used.

What is JavaScript object destructuring?

Object destructuring is the process of extracting multiple properties or items from objects at once. This technique is really useful when accessing object properties in JavaScript.

For example, suppose you have an object named **article** with multiple properties like in the following code example:



```
let article = {  
  id: '001',  
  title: 'JavaScript Object Destructuring',  
  writer: 'Chameera'  
};
```

Without object restructuring, you will have to extract the properties like this:



```
let id = article.id;  
let title = article.title;  
let writer = article.writer;
```

However, if you use object destructuring, you can extract all the properties with a single line of code:



```
let { id, title, writer } = article;  
  
console.log(id); // 001  
console.log(title); // JavaScript Object Destructuring  
console.log(writer); // Chameera
```

Here, the object properties are assigned to **articleId**, **articleName**, and **writerName**, respectively. If the variables have the same name as the object properties, you can write the code more concisely as follows:

```
let { id, title, writer } = article;

console.log(id); // 001
console.log(title); // JavaScript Object Destructuring
console.log(writer); // Chameera
```



We have seen how JavaScript object destructuring works, so let's discuss different techniques we can use.

1. Nested object destructuring

As developers, we often have to deal with nested objects. With the traditional approach, we have to follow the dot notation to access the nested properties.

```
let article = {
  id: '001',
  title: 'JavaScript Object Destructuring',
  writer: {
    name: 'Chameera'
  }
}
```



```
}  
};  
  
console.log(article.writer.name) // Chameera
```

However, object destructuring allows you to extract nested properties without using multiple destructuring assignments directly.

```
let article = {  
  id: '001',  
  title: 'JavaScript Object Destructuring',  
  writer: {  
    name: 'Chameera'  
  }  
};  
  
let {  
  id: articleId,  
  title: articleTitle,  
  writer: {  
    name: writerName  
  }  
} = article;  
  
console.log(articleId); // 001  
console.log(articleTitle); // JavaScript Object Destructuring  
console.log(writerName); // Chameera
```

 Copy

2. Destructuring function arguments

There are scenarios where we pass objects into functions and access the properties of those functions using dot notation like this:

```
let article = {
  title: 'JavaScript Object Destructuring',
  writer: 'Chameera'
};

let displayDetails = (article) => {
  console.log(article.title); // JavaScript Object Destructuring
};

displayDetails(article);
```

 Copy

With object destructuring, we can extract the object properties from the function signature.

```
let article = {
  title: 'JavaScript Object Destructuring',
  writer: 'Chameera'
};

let displayDetails = ({title, writer}) => {
```

 Copy

```
console.log(title); // JavaScript Object Destructuring
console.log(writer); // Chameera
};

displayDetails(article);
```

3. Destructuring with default values

Sometimes, the same types of objects might not have the exact number of properties. You can assign default values for the properties using object destructuring in such situations.

For example, suppose you are getting a list of articles from a database, and some articles do not have a subtitle. In such situations, you can assign a default value while destructuring. If the object contains a subtitle, the real value will be assigned to the variable. If not, default values will be assigned.

```
let article = {
  title: 'JavaScript Object Destructuring',
  writer: 'Chameera'
};

let { title, subtitle = 'No Sub Title', writer} = article;
```



```
console.log(title); // JavaScript Object Destructuring
console.log(subtitle); // No Sub Title
console.log(writer); // Chameera

let article = {
  title: 'JavaScript Object Destructuring',
  subtitle: 'Top 7 JavaScript Object Destructuring Techniques',
  writer: 'Chameera'
};

let { title, subtitle = 'No Sub Title' writer} = article;

console.log(title); // JavaScript Object Destructuring
console.log(subtitle); // Top 7 JavaScript Object Destructuring Techniques
console.log(writer); // Chameera
```

4. Destructuring null objects

In JavaScript, it is common to return objects from functions, and developers directly extract the properties of the returned object using object destructuring.

```
function getArticle() {

  let article = {
    title: 'JavaScript Object Destructuring',
    subtitle: 'Top 7 JavaScript Object Destructuring Techniques'
  };
}
```

[Copy](#)


```
    return article.  
  }  
  
  let { title, subtitle} = getArticle();
```

However, there can be scenarios where the returned object is null. In such situations, we need to modify the object destructuring statement to handle null objects. If not, the application will throw a **TypeError**.

```
function getArticle() {  
  return null.  
}  
  
let { title, subtitle} = getArticle(); // Type Error  
let { title, subtitle} = getArticle() || {}; // No Error
```

 Copy

5. Using the rest parameter

The rest parameter (...) allows functions to accept an indefinite number of arguments as an array.

```
function myFunction(a, b, ...theArgs) {  
  // ...  
}
```

 Copy

```
}
```

We can use this rest parameter in object destructuring as well. If there are many properties in an object, we can assign the essential values to variables and unpack all the remaining values to another object using the rest parameter.

```
let languages = {  
  language1 : 'Java',  
  language2 : 'JavaScript',  
  language3 : 'C',  
  language4 : 'Python',  
  language5 : 'C#',  
  language6 : 'C++',  
  language7 : 'Ruby'  
};  
  
let { language1, language2, ...otherLanguages } = languages;  
  
console.log(language1); // Java  
console.log(language2); // JavaScript  
console.log(otherLanguages) ;  
/*  
{ "language3": "C", "language4" : "Python", "language5": "C#", "language6": "C++",  
  "language7": "Ruby"  
} */
```



6. Assigning dynamic object keys

In all the previous examples, object keys were fixed. So, can we use object destructuring if the object keys are dynamic? Yes. We can assign object keys dynamically when destructuring.

For example, suppose you have an object of users like below, and you need to extract the designation based on the username dynamically.

```
let users = {  
  Paul: 'Software Engineer',  
  Mike: 'Senior Software Engineer',  
  George: 'Team Lead',  
  ...  
};
```

 Copy

In such situations, we can restructure the object using a variable as the object key to only fetch the required value.

```
let users = {  
  Paul: 'Software Engineer',  
  Mike: 'Senior Software Engineer',  
  George: 'Team Lead',
```

 Copy

```
...  
};  
  
let userKey = 'Mike';  
let { [userKey] : selectedUser } = users;  
  
console.log(selectedUser); // Senior Software Engineer
```

Likewise, you can change the value of the **userKey** variable and fetch the user details dynamically.

7. Combined object and array destructuring

JavaScript supports both object and array destructuring. If you have an array with multiple objects inside, you can combine the power of both object and array destructuring to extract nested objects and their properties easily.

```
let languages = [  
  { id: 1, name: 'Java' },  
  { id: 2, name: 'JavaScript' },  
  { id: 3, name: 'C' },  
  { id: 4, name: 'Python' },  
  { id: 5, name: 'C#' },  
  { id: 6, name: 'C++' },  
  { id: 7, name: 'Ruby' }  
];
```



```
let [, , , { name }, , , ] = languages;  
  
console.log(name) //Python
```

As you can see, I have used commas to skip the array objects. It is a feature in JavaScript array destructuring. You can use commas to skip elements of the array and only get the required elements.

Conclusion

In this article, I discussed seven JavaScript destructuring techniques every developer should know. However, the above list is only a subset of available object destructuring techniques.

If you haven't used object destructuring in JavaScript before, you can start with the techniques in this article and gradually move on to others. Adopting these techniques will improve your coding skills and make your code more readable.

I hope you found this article helpful. Thank you for reading.

Syncfusion [Essential JS 2](#) is the only suite you will ever need to build an app. It contains over 65 high-performance, lightweight, modular, and responsive UI

components in a single package. Download a [free trial](#) to evaluate the controls today.

If you have any questions or comments, you can also contact us through our [support forums](#), [support portal](#), or [feedback portal](#). We are always happy to assist you!

Related blogs

- [What's New in 2022 Volume 2: Essential JS 2](#)
- [Using Design Patterns in JavaScript —The Ultimate Guide](#)
- [Pure and Impure Functions in JavaScript: A Complete Guide](#)
- [Introducing License Key Validation for the Essential JS 2 Platforms](#)



MEET THE AUTHOR

Chameera Dulanga

Software engineer with more than 3 years of working experience in React, Angular, Node.js, Serverless, and AWS. Recognized as an AWS Community Builder for 2022. Tech Blogger since 2019 with 120+ articles.

