JavaScript Object Destructuring

Summary: in this tutorial, you'll learn about JavaScript object destructuring which assigns properties of an object to individual variables.

> If you want to learn how to destructure an array, you can check out the array destructuring tutorial.

Introduction to the JavaScript object destructuring assignment

Suppose you have a `person` object with two properties: `firstName` and `lastName`.

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};
```

Before ES6, when you want to assign properties of the `person` object to variables, you typically do it like this:

```
let firstName = person.firstName;
let lastName = person.lastName;
```

ES6 introduces the object destructuring syntax that provides an alternative way to assign properties of an object to variables:

```
let { firstName: fname, lastName: lname } = person;
```

In this example, the `firstName` and `lastName` properties are assigned to the `fName` and `lName` variables respectively.

In this syntax:

```
let { property1: variable1, property2: variable2 } = object;
```

The identifier before the colon (`:`) is the property of the object and the identifier after the colon is the variable.

> Notice that the property name is always on the left whether it's an object literal or object destructuring syntax.

If the variables have the same names as the properties of the object, you can make the code more concise as follows:

```
let { firstName, lastName } = person;

console.log(firstName); // 'John'
console.log(lastName); // 'Doe'
```

In this example, we declared two variables `firstName` and `lastName`, and assigned the properties of the person object to the variables in the same statement.

It's possible to separate the declaration and assignment. However, you must surround the variables in parentheses:

```
({firstName, lastName} = person);
```

If you don't use the parentheses, the JavaScript engine will interpret the left-hand side as a block and throw a syntax error.

When you assign a property that does not exist to a variable using the object destructuring, the variable is set to `undefined`. For example:

```
let { firstName, lastName, middleName } = person;
console.log(middleName); // undefined
```

In this example, the `middleName` property doesn't exist in the `person` object, therefore, the `middleName` variable is `undefined`.

Setting default values

You can assign a default value to the variable when the property of an object doesn't exist. For example:

```
let person = {
    firstName: 'John',
    lastName: 'Doe',
    currentAge: 28
};

let { firstName, lastName, middleName = '', currentAge: age = 18 } = person;

console.log(middleName); // ''
console.log(age); // 28
```

In this example, we assign an empty string to the `middleName` variable when the person object doesn't have the `middleName` property.

Also, we assign the `currentAge` property to the `age` variable with the default value of 18.

However, when the `person` object does have the `middleName` property, the assignment works as usual:

```
let person = {
    firstName: 'John',
    lastName: 'Doe',
    middleName: 'C.',
    currentAge: 28
};

let { firstName,  lastName, middleName = '', currentAge: age = 18 } = person;

console.log(middleName); // 'C.'
console.log(age); // 28
```

Destructuring a null object

A function may return an object or null in some situations. For example:

```
function getPerson() {
    return null;
}
```

And you use the object destructuring assignment:

```
let { firstName, lastName } = getPerson();

console.log(firstName, lastName);
```

The code will throw a `TypeError`:

```
TypeError: Cannot destructure property 'firstName' of 'getPerson(...)' as it is null.
```

To avoid this, you can use the `OR` operator (`||`) to fallback the `null` object to an empty object:

```
let { firstName, lastName } = getPerson() || {};
```

Now, no error will occur. And the `firstName` and `lastName` will be `undefined`.

Nested object destructuring

Assuming that you have an `employee` object which has a `name` object as the property:

```
let employee = {
    id: 1001,
    name: {
        firstName: 'John',
        lastName: 'Doe'
    }
};
```

The following statement destructures the properties of the nested `name` object into individual variables:

```
let {
    name: {
        firstName,
        lastName
    }
} = employee;

console.log(firstName); // John
console.log(lastName); // Doe
```

It's possible to do multiple assignment of a property to multiple variables:

```
let employee = {
    id: 1001,
    name: {
        firstName: 'John',
        lastName: 'Doe'
    }
};

let {
    name: {
        firstName,
        lastName
    },
    name
} = employee;

console.log(firstName); // John
console.log(lastName); // Doe
console.log(name); // { firstName: 'John', lastName: 'Doe' }
```

Destructuring function arguments

Suppose you have a function that displays the person object:

```
let display = (person) => console.log(`${person.firstName} ${person.lastName}`);

let person = {
    firstName: 'John',
    lastName: 'Doe'
};

display(person);
```

It's possible to destructure the object argument passed into the function like this:

```
let display = ({firstName, lastName}) => console.log(`${firstName} ${lastName}`);

let person = {
```

```
    firstName: 'John',
    lastName: 'Doe'
};

display(person);
```

It looks less verbose especially when you use many properties of the argument object. This technique is often used in React.

Summary

- Object destructuring assigns the properties of an object to variables with the same names by default.