JS Goodies: Nullish Coalescing and Optional Chaining

One of the things I like about the annual release schedule for Javascript is that the learning curve for new versions is much smaller than ES2015/ES6 and previous versions.

Two of the features that I'm most interested in are optional Chaining and Nullish Coalescing.

Optional Chaining

Optional Chaining, allows yout to check the existence of a property or short circuit and return `undefined` if any property in the chain that doesn't exist.

In the example below, we define a zoo object with animal types and their names.

```
const zoo = {
  name: 'Alice',
  bird: {
    name: 'Hawkeye',
  },
  dog: {
    name: 'Fluffy'
  },
  cat: {
    name: 'Dinah'
  }
};
```

We can then query for properties at any point down the chain. The dog object doesn't have a breed property so, if we use `zoo.dog?.breed;` to query for the breed, it will return undefined because the property doesn't exist rather than an

error as we'd normally expect.

```
const dogBreed = zoo.dog?.breed;
console.log(dogBreed);
// Outputs undefined

const dogName = zoo.dog?.name;
console.log(dogName);
// Outputs Fluffy

const birdType = zoo.bird?.type;
console.log(birdType);
// Outputs undefined
```

This makes it easier to query long chains of parent/child elements and avoid fatal errors in our applications.

Nullish coalescing operator

Nullish coalescing operator addresses an interesting shortcoming with the logical or operator || when it comes to setting up default values for an application.

```
const mySetting = '' || 'setting 1';
```

If the left-hand value can be converted to true then that's what the application will use, otherwise the value on the right-hand side will be used.

> These expression evaluate to false in Javascript:
>
> - null
> - NaN
> - 0
> - empty string (“” or ‘’ or ``)
> - undefined

But there's a problem with this method of setting values for preferences. There are times when an empty or otherwise false value (other than `null` or `undefined`) is acceptable for the setting that we want to work with.

That's where the `nullish coalescing operator` comes into play. It will produce the right side value if the left side value is null and the left value otherwise.

In the first example, the value of foo will be `default string` because the left side value is null. In this case the behavior is the same as the logical or operator.

```
const foo = null ?? 'default string';
console.log(foo);
// expected output: default string

const foo2 = null || 'default string';
console.log(foo2);
'default string'// expected output: default string
```

In the second example the value of baz will be 0. The first value is not null or undefined so the constant takes the left side value.

Compare the result with the baz2 constant where, using the logical or operator, we get the value of 42. 0 is a falsy value so we use the right side value as the value of the constat.

```
const baz = 0 ?? 42;
console.log(baz);
// expected output: 0

const baz2 = 0 || 42;
console.log(baz2);
// expected output: 42
```

The differences are subtle and can lead to annoying bugs when they don't produce the value you expect. It's up to you which one you use as long as you're ok with the results you get.