Rest/Spread

ES6 provides us with a new operator, ..., that is context dependent. Its purpose is to capture the rest of a data set, or to spread content of an existing structure.

Destructuring

Now that we understand using rest and spread we can work it into our destructuring. Remembering our array example from the destructuring chapter we wanted to extract the first couple animals from our list.

*const* animals = ['horse', 'dog', 'fish', 'cat', 'bird'];

*const* [firstAnimal, secondAnimal] = animals;

copy

If we want the remaining animals, using a rest pattern at the end of the assignment.

*const* [firstAnimal, secondAnimal, ...otherAnimals] = animals;

otherAnimals

// => ['fish', 'cat', 'bird']

copy

Objects

Pretty neat. Can we do the same with objects? Why yes we can. While object spread came about a few years after ES2015 it seems like a good idea to examine it here.

*const* person = {

firstName: 'Bob',

lastName: 'Marley',

email: 'bob@marley.com',

password: 'sekureP@ssw0rd9',

username: 'barley',

addresses: [

{

address: '1600 Pennsylvania Avenue',

city: 'Washington, D.C.',

zipcode: '20500',

},

{

address: '221B Baker St.',

city: 'London',

zipcode: 'WC2N 5DU',

}

],

createdAt: 1543945177623

};

copy

Grabbing firstName and lastName from person is easy, along with assigning all other properties to a 'catchall'.

*const* { firstName, lastName, ...attributes } = person;

copy

Using spread we can quickly make complete copies of objects or arrays.

*const* personCopy = { ...person };

copy

There are some limitations. The copy is shallow, so any complex or nested structures with objects references will still point to the same object. We'll use our person objects to demonstrate.

*const* personCopy = { ...person };

personCopy === person

// => false

personCopy.addresses === person.addresses

// => true