Gmail Bikash Adhikari <virtual.bikash@gmail.com>

Generator and Iterator in JavaScript

1 message

Bikash <bikash.caim@gmail.com>
To: virtual.bikash@gmail.com Sat, Mar 8, 2025 at 2:54 PM

Generators and iterators are powerful features in JavaScript that allow you to work with sequences of data in a more controlled and efficient way. They enable you to create custom iteration behavior, which is particularly useful for handling large datasets or complex algorithms.

Iterators

An iterator is an object that defines a sequence and potentially a return value upon its termination. Specifically, an iterator is any object that implements the `Iterator` protocol by having a `next()` method that returns an object with two properties:

- `value`: The next value[1] in the sequence.
- `done`: A boolean indicating whether the sequence has been exhausted.

Here's a basic example:

JavaScript

```
function createIterator(array) {
  let index = 0;
  return {
    next: function() {
      if (index < array.length) {
        return { value: array[index++], done: false };
      } else {
        return { value: undefined, done: true };
      }
    }
  };
}

const myArray = [1, 2, 3];
const myIterator = createIterator(myArray);

console.log(myIterator.next()); // { value: 1, done: false }
console.log(myIterator.next()); // { value: 2, done: false }
console.log(myIterator.next()); // { value: 3, done: false }
console.log(myIterator.next()); // { value: undefined, done: true }
```

Iterables

An iterable is an object that defines its iteration behavior, such as what values are looped over in a `for...of` construct. To be iterable, an object must implement the `Iterable` protocol, which means it must have a method with a `Symbol.iterator` key that returns an iterator.

Many built-in JavaScript objects are iterable, including arrays, strings, maps, and sets.

JavaScript

```
const myArray = [1, 2, 3];

for (const value of myArray) {
  console.log(value);
}
// Output:
// 1
// 2
// 3
```

Generators

Generators are a special type of function in JavaScript that can be paused and resumed. They provide a more convenient way to create iterators. Generators are defined using the `function*` syntax and use the `yield` keyword to produce values.

JavaScript

```
function* numberGenerator() {
  yield 1;
  yield 2;
  yield 3;
}

const generator = numberGenerator();

console.log(generator.next()); // { value: 1, done: false }
console.log(generator.next()); // { value: 2, done: false }
console.log(generator.next()); // { value: 3, done: false }
console.log(generator.next()); // { value: undefined, done: true }
```

Key features of generators:

- `yield` **keyword:** Pauses the generator's execution and returns the yielded value.
- `next()` **method:** Resumes the generator's execution until the next `yield` or the end of the function.
- **Lazy evaluation:** Values are produced on demand, which can be more efficient for large or infinite sequences.
- **Simplified iterator creation:** Generators make it easier to define custom iteration logic.
- `yield*` **keyword:** delegates to another iterable or generator.

Example with `yield*`:

JavaScript

```
function* anotherGenerator() {
  yield 4;
  yield 5;
}

function* mainGenerator() {
  yield 1;
  yield 2;
  yield* anotherGenerator();
  yield 6;
}

const mainGen = mainGenerator();

for (const value of mainGen) {
  console.log(value);
}
// Output:
// 1
// 2
// 4
// 5
// 6
```

Use cases:

- **Asynchronous programming:** Generators can simplify asynchronous code by making it look synchronous.
- **Lazy evaluation:** Generating values on demand, which is efficient for large datasets.
- **Custom iterators:** Creating complex iteration logic.
- **State machines:** Implementing stateful computations.
- **Infinite sequences:** Creating sequences that can produce an unlimited number of values.

Generators and iterators provide a powerful and flexible way to work with sequences of data in JavaScript. They promote cleaner, more efficient, and more readable code, especially when dealing with complex iteration scenarios.

Sources

1. http://boniplant.com/dmdudl/next-js-persist-state.html
2. https://juejin.cn/post/7205989194262216765
3. https://on-ad.github.io/javascript/2019/04/03/%EC%9E%90%EB%B0%94%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%8A%B8-%ED%95%A8%EC%88%98.html
4. https://juejin.cn/post/7264183910597279799
5. https://juejin.im/post/5d0c2f3ef265da1b8b2b6584
6. https://github.com/ishwarrimal/frontend-interview-preps
7. https://mockingbot.com/posts/244?page=1
8. https://velog.io/@junh0328/DEEP-DIVE-%ED%95%9C-%EC%9E%A5-%EC%9A%94%EC%95%BD-%EC%A0%9C%EB%84%88%EB%A0%88%EC%9D%B4%ED%84%B0-asyncawait