Generator

Intro

什么是 JavaScript Generators 呢？通俗的讲 Generators 是可以用来控制迭代器的函数。它们可以暂停，然后在任何时候恢复。如果这句话不好理解，可以看下接下来的示例。

1. 常规循环

```
for (let i = 0; i < 5; i += 1) {
  console.log(i)
}
// this will return immediately 0 -> 1 -> 2 -> 3 -> 4
```

2. 利用 Generator

```
function * generatorForLoop () {
  for (let i = 0; i < 5; i += 1) {
    yield console.log(i)
  }
}

const genForLoop = generatorForLoop()

console.log(genForLoop.next()) // first console.log - 0
console.log(genForLoop.next()) // 1
console.log(genForLoop.next()) // 2
console.log(genForLoop.next()) // 3
console.log(genForLoop.next()) // 4
```

对比下代码，常规的循环只能一次遍历完所有值，Generator 可以通过调用 next 方法拿到依次遍历的值，让遍历的执行变得“可控”。

Basic Syntax

语法

```
function * gen () {
  yield 1
  yield 2
  yield 3
}

let g = gen()
// "Generator { }"
```

这个是 Generator 的定义方法，有几个点值得注意：

1. 比普通函数多一个 *
2. 函数内部用 yield 来控制程序的执行的“暂停”
3. 函数的返回值通过调用 next 来“恢复”程序执行

> [!DANGER]
> Generator 函数的定义不能使用箭头函数，否则会触发 SyntaxError 错误

```
let generator = *() => {} // SyntaxError
let generator = ()* => {} // SyntaxError
let generator = (*) => {} // SyntaxError
```

这些做法都是错误的✕。