ES6 Generators

...

@jussinieminen

Generators are function executions that can be suspended and resumed later

Normal function

```
function myfunc(){
 var x = 1;
 var y = 3;
 return x * y;
}

console.log(myfunc()); // 3
```

step1.js

Generator

```
function *generatorFunction(){
 console.log("Hello World!");
}

generatorFunction();
```

step2.js

Executing generator

```
function *generatorFunction(){
 console.log("Hello World!");
 var x = 1;
 var y = 3;
 yield;
 console.log(x * y);
}

var iter = generatorFunction();
iter.next(); // Hello World!
iter.next(); // 3
```

step3.js

Parameters in

```
function *generatorFunction(x){
 var y = yield;
 var result =  x * y;
 console.log(result);
}

var iter = generatorFunction(2);
iter.next();
iter.next(3); // 6
```

step4.js

Parameters out

```
function *generatorFunction(x){
 var y = yield "Hello World!";
 var result =  x * y;
 return result;
}

var iter = generatorFunction(2);
var fromYield = iter.next();
console.log(fromYield); // { value: 'Hello World!', done: false }
var result = iter.next(3);
console.log(result); // { value: 6, done: true }
```

step5.js

Generator delegation

```
function *gen1(){
 console.log("gen1");
 yield 1;
 yield *gen2();
 yield 3;
}

function *gen2(){
 console.log("gen2");
 yield 2;
}

var iter = gen1();
console.log(iter.next().value);     // gen1
                                    // 1
console.log(iter.next().value);     // gen2
                                    // 2
console.log(iter.next().value);     // 3
```

step7.js

Async functions

```
function somethingAsync(callback){
 setTimeout(function(){
   var data = "Hello World from async function";
   callback(data);
 }, 1000);
}

somethingAsync(function(data){
 console.log(data)
});
```

step8.js

Would it be nice...

```
function somethingAsync(){
 setTimeout(function(){
   return "Hello World from async function";
 }, 1000);
}

var result = somethingAsync();
console.log(result); //undefined
// exits after 1000ms
```

step9.js

We can… almost

```
function somethingAsync(){
 setTimeout(function(){
   iter.next("Hello World from async function!");
 }, 1000);
}

function *run(){
 var result = yield somethingAsync();
 console.log(result); // Hello World from async function!
}

var iter = run();
iter.next();
```

step10.js

What about Promises

```
function somethingAsync(){
 return new Promise(function (resolve, reject) {
   setTimeout(function () { resolve("Hello World from resolved promise!"); }, 1000);
 });
}

function *run(){
 var result = yield somethingAsync();
 console.log(result); // Hello World from resolved promise!
}

var iter = run();
var promise = iter.next().value;
promise.then(
 function(data){
   iter.next( data );
 },function(err){}
);
```

step11.js

Extract boilerplate

```
function run(gen) {
 var args = [].slice.call( arguments, 1), it;
 it = gen.apply( this, args );
 return Promise.resolve()
   .then( function handleNext(value){
     var next = it.next( value );
     return (function handleResult(next){
       if (next.done) {
         return next.value;
       }
       else {
         return Promise.resolve( next.value )
           .then(
             handleNext,
             function handleErr(err) {
               return Promise.resolve(
                 it.throw( err )
                 )
                 .then( handleResult );
             }
           );
       }
     })(next);
   } );
}
```

https://github.com/getify/You-Dont-Know-JS/blob/master/async%20&%20performance/ch4.md#promise-aware-generator-runner

With external runner

```
var ASQ = require("asynquence-contrib");

function somethingAsync(){
 return new Promise(function (resolve, reject) {
   setTimeout(function () { resolve("Hello World from resolved
promise!"); }, 1000);
 });
}

ASQ().runner(function *main(){
 var result = yield somethingAsync();
 console.log(result); // Hello World from resolved promise!
});
```

step12.js

Full flow control

```
function helloAsync(){
 return new Promise(function (resolve, reject) {
   setTimeout(function () { resolve("Hello"); }, 500);
 });
}

function worldAsync(){
 return new Promise(function (resolve, reject) {
   setTimeout(function () { resolve("World"); }, 200);
 });
}

ASQ().runner(function *main(){
 var hello = yield helloAsync();
 console.log(hello); // Hello
 var world = yield worldAsync();
 console.log(hello + " " + world); // Hello World
});
```

step13.js

asynquence library

https://github.com/getify/asynquence by Kyle Simpson

Future

async & await

```
async function main(){
 var hello = await helloAsync();
 var world = await worldAsync();
 console.log(hello + " " + world); // Hello World
}
```

Other usage

- Lazy evaluation
- infinite sequences

Thank You

@jussinieminen

https://github.com/jniemin/ES6-generators-presentation