

Các syntax quan trọng trong ES6/7 cho React / React Native



VIET TRAN · CHỦ NHẬT, 11 THÁNG 6, 2017 · READING TIME: 9 MINUTES

Javascript (EcmaScript - ES) hiện tại là ngôn ngữ phổ biến nhất bởi tính linh động được sử dụng cho cả Website Frontend, Backend cho tới mobile. Trong những năm qua, cộng đồng JS đã phát triển không ngừng nghỉ dẫn tới việc ra đời ES6, 7 và gần đây là ES8 gần như liên tiếp qua mỗi năm. Đây là một tốc độ khá là chóng mặt cho các tín đồ JS theo.

React Native sử dụng ES6/7 để các developers viết ứng dụng trên nó. Tuy nhiên Facebook chỉ lấy 1 phần (subset) của ES6/7 và sử dụng Babel để làm transpiler. Chi tiết các bạn có thể xem tại [đây](#).

Dưới đây là các plugin ES6/7 mà React Native đang sử dụng và bài viết này minh tập trung vào những syntax mà mình thấy quan trọng nhất trong React Native.

```
'syntax-async-functions',
'syntax-class-properties',
'syntax-trailing-function-commas',
'transform-class-properties',
'transform-es2015-function-name',
'transform-es2015-arrow-functions',
'transform-es2015-block-scoping',
'transform-es2015-classes',
'transform-es2015-computed-properties',
'check-es2015-constants',
'transform-es2015-destructuring',
['transform-es2015-modules-commonjs', { strict: false, allowTopLevelThis: true }],
'transform-es2015-parameters',
'transform-es2015-shorthand-properties',
'transform-es2015-spread',
'transform-es2015-template-literals',
'transform-es2015-literals',
'transform-flow-strip-types',
'transform-object-assign',
'transform-object-rest-spread',
'transform-react-display-name',
'transform-react-jsx',
'transform-regenerator',
['transform-es2015-for-of', { loose: true }],
require('../transforms/transform-symbol-member')
```

Khai báo biến với `var`, `let`, và `const`

Khi khai báo biến với Const, biến đó sẽ là immutable variable, nghĩa là sẽ không thay đổi được giá trị của biến.

Với var và let, chúng ta đều có thể khai báo được 1 biến bất kỳ, biến này có thể thay đổi được giá trị. Điểm khác biệt giữ var và let đó là:

```
// var cho phép chúng ta khai báo lại 1 biến cũ, nhưng let thì không
var n = 1;
var n = 2; // no syntax error

let m = 1;
let m = 2; // syntax error

// var và let đều tác động vào function block như nhau, tuy nhiên ở trường hợp này let sẽ chỉ tác động vào block ngay sau nó:
```

```
function someFunc() {

    for( let i = 0; i <= 9; i++ ) {
        // Biến i lúc này chỉ tồn tại trong scope block của for
    }

    // Gọi biến i ngoài này sẽ bị lỗi
}

function someFunc() {

    for( var i = 0; i <= 9; i++ ) {
        // Biến i lúc này không chỉ tồn tại trong scope block của for
    }

    // mà còn tồn tại cả ở ngoài này nữa, lúc này biến i = 10
}
```

Từ đó chúng ta có thể thấy trong các ứng dụng RN sẽ ưu tiên sử dụng let nhiều hơn để sourcecode chặt chẽ hơn và không bị các lỗi không mong muốn.

Arrow function

Basic syntax của Arrow function:

```
(param1, param2, ..., paramN) => { statements }
(param1, param2, ..., paramN) => expression
// equivalent to: (param1, param2, ..., paramN) => { return expression; }
```

Arrow function cũng như function bình thường, chỉ khác về syntax và binding context:

```
function Pet() {
    this.age = 1;

    function showAgeFunc() {
        alert( 'Age in showAgeFunc: ' + this.age ); // Age in showAgeFunc: undefined
    }
}
```

```

const showAgeArrowFunc = () => {
  alert( 'Age in showAgeArrowFunc: ' + this.age ); // Age in showAgeFunc: 1
}

setTimeout( showAgeFunc, 1000);
setTimeout( showAgeArrowFunc, 1000);
}

new Pet();

```

Ngoài ra các bạn có thể xem thêm arrow function tại [đây](#).

Module import / export

Ứng dụng RN thường được phát triển trên nhiều file JS mà ta thường gọi là các module / component. Tất cả các biến và function trong 1 file JS sẽ chỉ được truy xuất trong file (hay còn gọi là file private). Để cho phép các thành phần có thể sử dụng từ các file khác, chúng ta cần dùng tới từ khoá export và import.

```

//// File CurrencyConverter.js

const rate = 22650.0; // Private in file

// export for public use
export const bankName = 'ACB';
export const vnd2dollar = (vnd) => vnd / rate;

// export mặc định
export default dollar2vnd = (dolla) => dolla * rate;

//// File Other.js

import Convert from './CurrencyConverter';

// Convert chính là hàm dollar2vnd được export mặc định
alert(Convert(10)); // 226500

//// File Another.js
import Convert, { vnd2dollar, bankName } from './CurrencyConverter';

// Import thêm vnd2dollar và bankName trong file CurrencyConverter.js
alert(Convert(10)); // 226500
alert(vnd2dollar(226500)); // 10

```

Function Parameter: default và rest

```

// url và method có giá trị mặc định nếu không được truyền vào khi gọi function.
const requestURL = (url = '', method = 'GET') => {

};

```

```

requestURL(); // url = '', method = 'GET'
requestURL('http://facebook.com'); // url = 'http://facebook.com', method = 'GET'

const requestURLWithParams = (url = '', method = 'GET', ...params) => {
  // params là array chứa giá trị các tham số thứ 3 trở đi
}

requestURLWithParams(); // params = []
requestURLWithParams('someURL','GET', 'a', 1, {}); // params = ['a', 1, {}]

```

Object/Array Matching, Short Hand Notation

```

let arr = [1,2,3];
let [n, , m] = arr; // n = 1, m = 3
console.log(n, m); // 1, 3

let person = { name : 'Viet Tran', age: 28, interestedIn: 'Coding' };
let { name, age } = person; // name = 'Viet Tran', age = 28
console.log(name, age); // 'Viet Tran', 28

let { name , job = 'some job' } = person; // nếu person không có thuộc tính job thì job
= 'some job' như một giá trị mặc định

```

Spread Operator

Đây là một trong những operator quan trọng chúng ta rất hay dùng trong RN. Trong clip hướng dẫn RN cơ bản mình cũng có demo quản lý style component con với Spread Operator.

```

let arr = [1,2,3];
let someArr = [...arr, 4]; // [1,2,3,4], ...arr gọi là spread operator
let thatArr = [4, ...arr]; // [4,1,2,3], tương tự nhưng chỉ khác vị trí các thành phần.

let p1 = { x: 0, y: 1 };
let p2 = { ...p1, z: 1}; // { x: 0, y: 1, z: 1 }
let p3 = { ...p2, y : 2 } // { x: 0, y: 2, z: 1 }, update y nếu y đã có.
let p4 = { y : 3, ...p3 } // { y : 2, x: 0, z: 1}, không update y vì nguyên tắc phía
sau override phía trước

```

String interpolation

Đây là một trong những tính năng của ES6 mà mình rất hay sử dụng.

```

const number = 10;
const str = `number = ${number}`; // đây là cách cũ thường được dùng trong ES5

```

```
const str = `number = ${ number }` // đây là cách mới trong ES6
const str = `number + 1 = ${ number + 1} // chúng ta có thể sử dụng expression trong
'{}'

const str = `Some string`; // chuỗi bình thường cũng xài được
```

Classes

Class rất quan trọng trong RN, đây là kỹ thuật chính chúng ta sử dụng để xây dựng các Component. Phân này có liên quan tới OOP khá nhiều nên mình chỉ nói các kỹ thuật chính.

```
class SomeObject {}
let obj = new SomeObject(); // tạo biến obj là đối tượng của SomeObject

// Các biến và hàm trong class không cần khai báo với từ khoá var/let/constant/function

class Pet {

    // biến trong class
    food = `something eatable`;

    // Hàm (method) trong class
    eat() {
        console.log('I can eat ${ this.food }');
    }

    // Hàm khởi tạo (constructor) cho class Pet
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
}

let myPet = new Pet('Beo', 2);
console.log(myPet); // object { food: 'something eatable', name: 'Beo', age: 1 }
myPet.eat(); // I can eat something eatable

// Khai báo class Cat kế thừa từ class Pet
class Cat extends Pet {

    static numberOfLegs = 4; // biến static trong class

    // Hàm static trong class
    static lazy() {
        console.log(`All cats are lazy`);
    }

    constructor(name, age) {
        super(name, age); // gọi lên hàm dựng của parent class: Pet
        this.food = `fishes`;
    }
}

let myCat = new Cat();
myCat.eat(); // I can eat fishes. Hàm eat được kế thừa từ class cha (Pet)
console.log(Cat.numberOfLegs); // 4
Cat.lazy(); // All cats are lazy
```

Promise và parallel promise

Việc sử dụng Promise là một giải pháp hiệu quả khi làm việc với các hàm callback, sourcecode chúng ta sẽ dễ đọc hơn.

```
const doSomething = (err) => {
  return new Promise( (resolve, reject) => {
    setTimeout( () => {
      if (err) {
        reject(err);
      } else {
        resolve(10);
      }
    }, 1000);
  });
}

doSomething().then( result => {
  console.log(result); // print `10` after 1s
}).catch(err => {
  console.log(err);
});

doSomething(`Something went wrong`).then( result => {
  console.log(result);
}).catch(err => {
  console.log(err); // print `Something went wrong` after 1s
});
```

Chúng ta có thể sử dụng hàm `then` như một middleware, ở mỗi bước `then` ta có thể return để làm tham số cho hàm `then` tiếp theo.

```
doSomething().then( result => {
  console.log(result); // 10
  return result * 2;
}).then( result => {
  console.log(result); // 20
  return { n: result }
}).then( result => {
  console.log(result); // { n : 20 }
})
.catch(err => {
  console.log(err);
});
```

Chạy cùng lúc nhiều Promise với Promise.all. Việc này rất hiệu quả khi ta cần load 1 lúc nhiều APIs hoặc nhiều async tasks song song.

```

const doSomething = (result, timeout) => {
  console.log(result, timeout);
  return new Promise( (resolve, reject) => {
    setTimeout( () => {
      resolve(result);
    }, timeout);
  });
}

Promise.all([
  doSomething(`OK`, 2000),
  doSomething(100, 5000)
]).then( data => {
  console.log(data); // print ['OK', 100] after 5s

  let [ first, second] = data;
  console.log(first, second); // 'OK', 100

}, err => {
  console.log(er);
});

```

Async và Await

Thế giới của JS đã`y rầy những callback function và promise, thế nhưng lắm lúc ta lại câ`n chúng chạy synchronize bình thường, hay nói đúng hơn ta săn sàng đợi cho chúng chạy xong. Source code sẽ chạy lâ`n lượt từ trên xuô`ng dưới.

```

async function doTask() { // có thể dùng `aync () => {}` để thay thế
  let result = await doSomething(100, 3000);
  console.log(result); // print 100 after 3s

  let nextResult = await doSomething(20, 2500);
  console.log(nextResult) // print 20 after 5.5s
}

doTask();
console.log(`Run here first`); // dòng này in trước khi doTask() chạy

```

Ở đây ta câ`n lưu ý: Từ khoá wait chỉ chạy trong function được khai báo với từ khoá async. Vì function này async nên sẽ chạy bâ`t đô`ng bộ (chạy ở 1 thread khác) nên ở trên ta thâ`y `Run here first` sẽ được in ra đâ`u tiên. Trong function doTask, từ khoá await sẽ khiê`n doSomething chạy như synchronize (block thread hiện tại để đợi kêt quả).

Kê`t

Việc nắm được các syntax quan trọng trong ES6/7 sẽ hỗ trợ rất nhiê`u cho chúng ta dev ứng dụng trên React/React Native, source code sẽ dễ đọc và maintain hơn. ES6/7 vẫn còn rất nhiều các tính năng hữu ích, trong một bài vié`t mình không thể liệt kê ra hê`t được. Các bạn có thể xem thêm tại:

<http://es6-features.org/#Constants>

<https://github.com/ericdouglas/ES6-Learning>

<https://leanpub.com/understandinges6>

98

18 bình luận 10 lượt chia sẻ

-
-  **Khanh Hung Nguyen** Hay quá man :)))
2 năm 1
-  **Nguyễn Vũ Hoàng** đã trả lời · 3 phản hồi
-  **Lâm Phước Thiện** vậy là async await chỉ khiến function async chạy synchronize chứ vẫn ko synchronize ở ngoài đúng ko a
2 năm 1
-  **Linh Đức Nguyễn** đã trả lời · 2 phản hồi
-  **Nhat Nguyen** Dc share rộng rãi ko Việt
2 năm 1
-  **Luu Vinh Tuong** xuất sắc, thanks Việt
2 năm · Đã chỉnh sửa 1
-  **Anh Tuan** Đại ca kết bạn với em đi
2 năm · Đã chỉnh sửa 1
-  **Cao Phuoc Thanh** Theo em bik thì arrow function và function bình thường có chút khác biệt.
2 năm
-  **Viet Tran** đã trả lời · 1 phản hồi
-  **Văn Trần** cảm ơn anh nhiều
2 năm
-  **Ngọc Danh** Cám ơn bạn đã chia sẻ
2 năm