**CLASS - OBJECT ES6**

**OBJECT**

**Đối tượng (Object) có thể chứa nhiều giá trị :**

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

**Đối tượng cũng có thể chứa phương thức hàm – function :**

var person = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};

**Truy cập thuộc tính đối tượng, có 2 cách :**

* *objectName.propertyName :* person.lastName;
* *objectName["propertyName"] :* person["lastName"];

**Truy cập hàm phương thức đối tượng :**

* *objectName.methodName() :* name = person.fullName();

**This – từ khóa trong đối tượng : là đại diện cho chính nó .**

var **person** = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  fullName : function() {  
    return **this**.firstName + " " + **this**.lastName;  
  }  
};

**CLASS**

**Đây là cách tạo đối tượng cũ trong javascipt :**

var employee = {

name: null,

age: null,

setName: function (name) {

this.name = name;

},

getName: function () {

return this.name;

},

setAge: function (age) {

this.age = age;

},

getAge: function () {

return this.age;

}

};

**Từ ES6 , đối tượng có thể tạo bằng cách này :**

class Employee {

setName (name) {

this.name = name;

}

getName () {

return this.name;

}

setAge (age) {

this.age = age;

}

getAge () {

return this.age;

}

constructor(name,age,wifeName)

{

super(name,age);

this.wifeName = wifeName;

}

};

Và với ES6, nó cũng hỗ trợ chúng ta một phương thức đặc biệt mà bất kỳ ngôn ngữ lập trình nào cũng có đối với class đó là constructor - phương thức khởi tạo. constructor trong ES6 cũng có tác dụng tương tự, nó sẽ tự động được gọi khi đối tượng được khởi tạo. Để khai báo constructor trong ES6 thì các bạn chỉ cần khai báo một phương thức có tên là **constructor**.

-------------------------------------------------------------------------------------

**MAP**

Map, mảng kết hợp (associate arrays) hay từ điển (dictionary/dict) là những thuật ngữ dùng để chỉ một cấu trúc dữ liệu, cho phép bạn ánh xạ từ một khóa (key) tương ứng với một giá trị (value). Trong JavaScript, chúng ta có thể sử dụng object để thể hiện cấu trúc này.

const dict = {

hello: 'Xin chào',

bye: 'Tạm biệt',

}

console.log(dict['hello']) // Xin chào

Tuy nhiên, nếu dùng object thì bạn chỉ có thể dùng chuỗi làm khóa. Ngoài ra, cách này cũng có một số hạn chế khác. Lớp Map do ES6 giới thiệu sẽ giúp giải quyết những vấn đề này. Với Map, bạn có thể sử dụng bất cứ dạng dữ liệu nào để làm khóa.

const obj = { bar: 2 }

const dict = new Map()

dict

.set('foo', 123)

.set(obj, 'hello world')

dict.get('foo') // 123

dict.get(obj) // 'hello world'

// Lấy giá trị của một khóa không tồn tại

dict.get('wat') // undefined

Bạn cũng có thể truyền vào hàm dựng của Map một mảng các cặp giá trị dạng [key, value], ví dụ như sau:

const dict = new Map([

['foo', 123],

[obj, 'hello world']

])

Như đã nói ở trên, bạn có thể dùng bất cứ dạng dữ liệu gì để làm khóa cho Map, kể cả mảng, object, hàm, hay NaN.

const arr = [1]

const f = () => {}

dict

.set(arr, 'an array')

.set(f, 'a function')

.set(NaN, 'not a number')

Nếu trong map đã có sẵn *khóa*, dữ liệu mới sẽ bị ghi đè lên.

const m = new Map()

m.set('foo', 1)

m.set('foo', 2)

m.get('foo') // 2

Để duyệt qua các *khóa* và giá trị trong Map, bạn có thể dùng:

const dict = new Map([

['foo', 1], ['bar', 2]

])

dict.keys() // ['foo', 'bar']

dict.values() // [1, 2]

dict.entries() // [ ['foo', 1], ['bar', 2] ]

dict.forEach(function(value, key, map) {

console.log(`${key} has ${value}`)

}, /\* thisArgs bạn có thể truyền vào tham chiếu cho `this` ở đây \*/)

// Sử dụng for..of cùng với destructuring

for (let [key, value] of dict) {

console.log(`${key} has ${value}`)

}

Bạn cũng có thể dùng toán tử spread ... với Map

const dict = new Map([

['foo', 1], ['bar', 2]

])

console.log([

['wut', 3],

...dict

])

// [ [ 'wut', 3 ], [ 'foo', 1 ], [ 'bar', 2 ] ]

Một số thao tác khác với Map.

const dict = new Map([

['foo', 1], ['bar', 2]

])

// Đếm số cặp giá trị trong map

dict.size // 2

// Kiểm tra trong map có khóa "foo" hay không

dict.has('foo') // true

dict.has('wut') // false

// Xóa một khóa, trả về boolean nếu thành công, false nếu thất bại

dict.delete('wut') // false

dict.delete('foo') // true

// Xóa hết các cặp giá trị của map

dict.clear()

Trong React, chúng ta điều khiển component bằng cách sử dụng **props và state.**

-------------------------------------------------------------------------------------

**PROPS**

* props là viết tắt của Properties.
* Một điều mà bạn cần phải nhớ khi sử dụng props đó là không bao giờ nên thay đổi giá trị của nó, hay nói cách khác, đây là một dữ liệu immutable.
* Các component nhận **props** từ component cha. Bạn không được thay đổi giá trị của **props** trong các component này mà chỉ được phép đọc giá trị ra thôi.
* Trong React thì dữ liệu sẽ đi theo một chiều, có nghĩa là từ component cha => các component con.
* Bạn có thể tạo ra component sử dụng props. Ý tưởng của props đó là việc trừu tượng hoá các component để có thể sử dụng được ở nhiều chỗ khác nhau trong app.

import React, { Component } from 'react';

import { AppRegistry, Text, View } from 'react-native';

class Greeting extends Component {

render() {

return (

<View style={{alignItems: 'center'}}>

<Text>Hello {this.**props**.name}!</Text>

</View>

);

}

}

export default class LotsOfGreetings extends Component {

render() {

return (

<View style={{alignItems: 'center'}}>

<Greeting name='Rexxar' />

<Greeting name='Jaina' />

<Greeting name='Valeera' />

</View>

);

}

}

-------------------------------------------------------------------------------------

**STATE**

* state thì hoạt động khác với props. state là dữ liệu nội bộ của một Component, trong khi props là dữ liệu được truyền cho Component.
* Chính vì vậy hoàn toàn có thể thay đổi state, và coi nó là một kiểu dữ liệu **mutable**.
* Tuy vậy, đừng bao giờ thay đổi trực tiếp biến this.state. Thay vào đó hãy dùng hàm setState để cập nhật giá trị.
* Sở dĩ cần dùng hàm này là do nó sẽ kích hoạt việc **render** lại component và tất cả component con nằm trong nó, còn thay đổi this.state thì không.
* **setState** chạy bất đồng bộ, vậy nên nếu tiến hành đọc ra giá trị state ngay **sau khi setState** thì chưa chắc giá trị sẽ được update .

Một ví dụ thiết lập hàm **setState** sẽ ẩn - hiện văn bản sau 1s thời gian :

class Blink extends Component {

constructor(props) {

super(props);

**this.state = { isShowingText: true };**

setInterval(() => ( // Toggle the state every second

**this.setState**(previousState => (

{ isShowingText: !previousState.isShowingText }

))

), 1000);

}

render() {

if (!this.**state**.isShowingText) {

return null;

}

return (

<Text>{this.**props**.text}</Text>

);

}

}

// Class chứa hiển thị từ class - **Blink**

export default class BlinkApp extends Component {

render() {

return (

<View>

<Blink text='I love to blink' />

<Blink text='Yes blinking is so great' />

</View>

);

}

}

-------------------------------------------------------------------------------------

**LISTVIEW**

ListView - Một thành phần cốt lõi được thiết kế để hiển thị hiệu quả các danh sách cuộn theo chiều dọc của dữ liệu thay đổi. API tối thiểu là tạo một [ListView.DataSource](https://facebook.github.io/react-native/docs/listviewdatasource), điền vào nó một mảng các đốm dữ liệu đơn giản và khởi tạo một ListViewthành phần với nguồn dữ liệu đó và một renderRowcuộc gọi lại lấy một đốm từ mảng dữ liệu và trả về một thành phần có thể kết xuất được.

class MyComponent extends Component {

constructor() {

super();

const ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});

this.state = {

dataSource: ds.cloneWithRows(['row 1', 'row 2']),

};

}

render() {

return (

<ListView

dataSource={this.state.dataSource}

renderRow={(rowData) => <Text>{rowData}</Text>}

/>

);

}

}

ListView cũng hỗ trợ các tính năng nâng cao hơn, bao gồm các phần có tiêu đề phần dính, hỗ trợ đầu trang và chân trang, gọi lại để đạt đến cuối dữ liệu có sẵn ( onEndReached) và trên tập hợp các hàng có thể nhìn thấy trong thay đổi chế độ xem thiết bị ( onChangeVisibleRows) và một số hiệu suất tối ưu hóa.

Có một vài thao tác hiệu suất được thiết kế để giúp ListView cuộn trơn tru trong khi tải động các bộ dữ liệu có khả năng rất lớn (hoặc vô hạn về mặt khái niệm):

* Chỉ kết xuất lại các hàng đã thay đổi - hàm rowHasChanged được cung cấp cho nguồn dữ liệu cho ListView biết nếu nó cần kết xuất lại một hàng vì dữ liệu nguồn đã thay đổi - xem ListViewDataSource để biết thêm chi tiết.
* Kết xuất hàng giới hạn tỷ lệ - Theo mặc định, chỉ có một hàng được hiển thị trên mỗi vòng lặp sự kiện (có thể tùy chỉnh với pageSizeprop). Điều này chia công việc thành các phần nhỏ hơn để giảm cơ hội rơi khung trong khi kết xuất hàng.

Sử dụng một trong các thành phần danh sách mới, chẳng hạn như [FlatList](https://facebook.github.io/react-native/docs/flatlist)hoặc [SectionList](https://facebook.github.io/react-native/docs/sectionlist)để sử dụng bộ nhớ bị ràng buộc, ít lỗi hơn, hiệu suất tốt hơn, API dễ sử dụng hơn và nhiều tính năng hơn .