ReactJS

* 1. Định nghĩa props, state, lifecycle
* Props

Thông tin truyền từ compponent này sang component khác được gọi là prop. Props của 1 component là 1 object để sữ dụng được thì ta phải dùng this.props



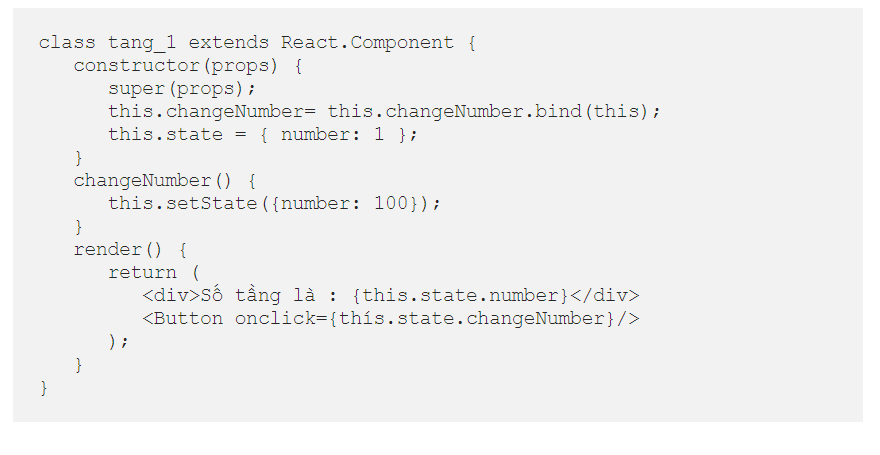


Bên cạnh đó thuộc tính this.props.children sẽ trã về mọi thứ nằm giữa các thẻ JSX đóng và mở

* State

Không giống như props thì State sẽ không thể truyền ra đc bên ngoài mà nó sẽ đc Component quản lí. Mỗi Component sẽ giữ riêng cho mình 1 State

Khai báo 1 State cần phải có 1 Contructor. Và theo 1 cấu trúc như bên dưới



State chỉ thay đổi khi ta sữ dụng thuộc tính this.setState để thay đổi

* Lifecycle Methods

Lifecycle là những phương thức được viết sẵn truy cập đến từng thời điểm trong một vòng đời của component, từ đó chúng ta có thể triển khai ý tưởng một cách dễ dàng linh hoạt, thuận lời cho quá trình fix bug sau này

* Chi tiết về vòng đời như sau:

1. **componentWillMount**: đây là method sẽ được thực thi trước khi 1 component được render.
2. r**ender**: Hàm hiển thị
3. **componentDidMount**: được gọi sau khi thực hiện update ham render.

* Có 5 phương thức được định nghĩa để thực hiện khi một phần tử được cập nhật:

1. **componentWillReceiveProps:**sẽ được thực thi ngay khi thuộc tính props được update và trước khi component được render lại.
2. **shouldComponentUpdate**sẽ trả về kết quả true or false. Phương thức này sẽ xác định 1 component có được update hay không. Mặc định giá trị này là true. Nếu bạn không muốn component render lại sau khi update state hay props thì return giá trị thành false.
3. **componentWillUpdate**được gọi khi chúng ta update state, props của component trước khi nó render lại.
4. **componentDidUpdate**sau khi componentWillUpdate ở trên được gọi xong thì đến lượt thằng này được goi.

Cuối cùng là:

**componentWillUnmount**: được gọi trước khi một component được xóa khỏi DOM

* 1. Các dạng Conditional Rendering

Trong Reactjs hỗ trợ các loại toán tử if else, switch case, condition ? true : flase, ngoài rea nó còn hỗ trợ thêm toán tử {condition && true}.

Cách sữ dụng {condition && true} như sau:



* 1. Khái niệm Hooks trong ReactJS

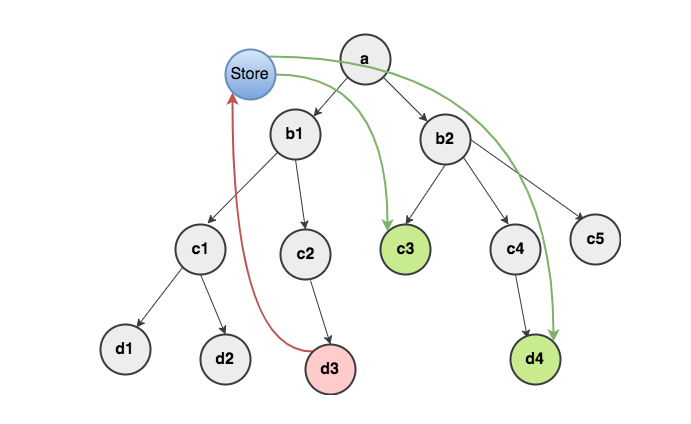
**Hooks** là những hàm cho phép bạn “kết nối” React state và lifecycle vào các components sử dụng hàm. Với Hooks bạn có thể sử dụng state và lifecycles mà không cần dùng ES6 class. Hooks chỉ có thể được gọi từ các function component, Hooks nên được gọi ở cấp cao nhất, không nên gọi trong các vòng lặp, điều kiện hoặc các hàm lồng nhau

**Tìm hiểu chi tiết tại:** <https://viblo.asia/p/tim-hieu-ve-react-hooks-jvElaGkDKkw>

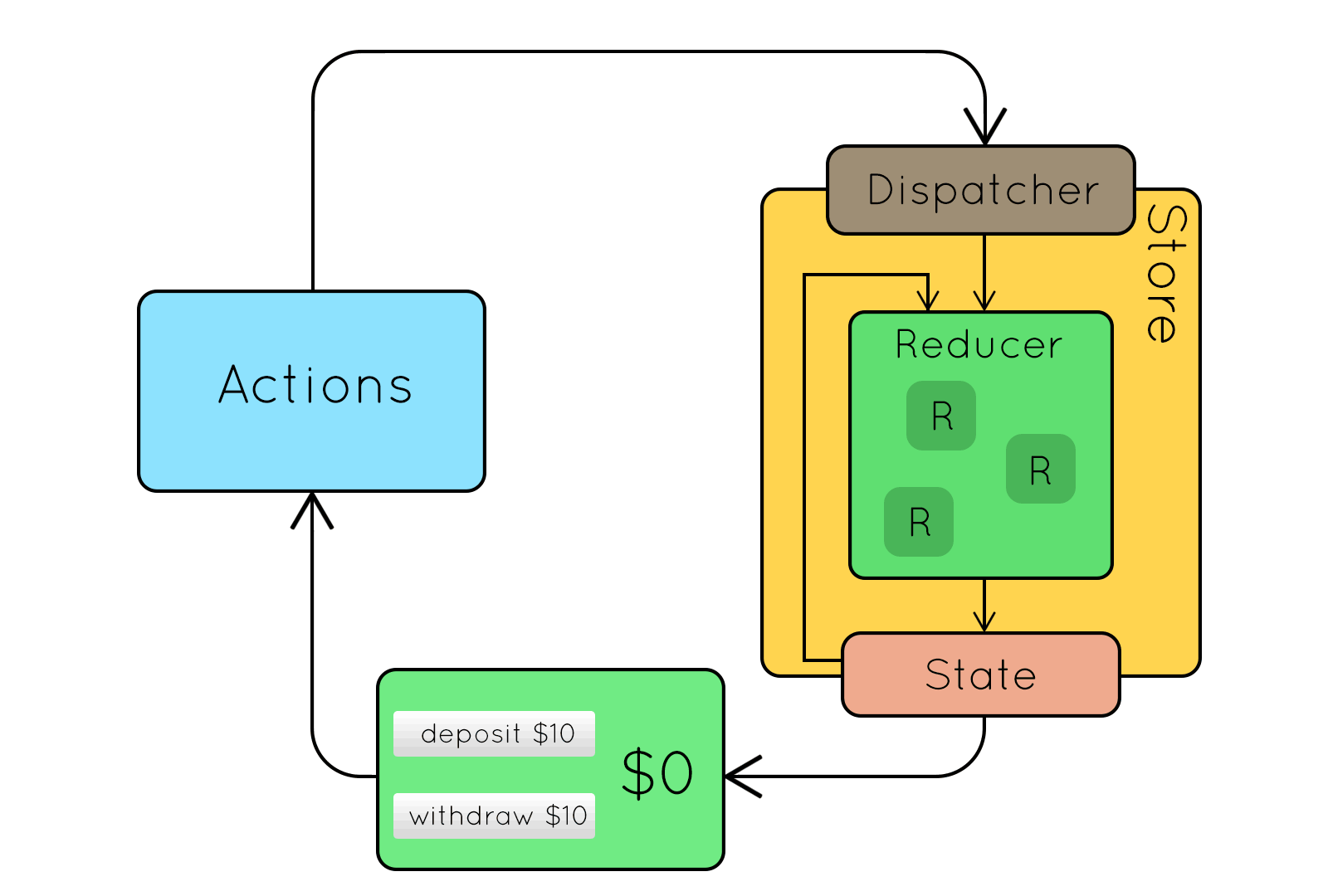
REDUX

Tóm gọn nội dụng redux <https://viblo.asia/p/redux-co-ban-m68Z00JdZkG>

* 1. Khái niệm cơ bản trong redux
* State toàn bộ ứng dụng được lưu trong 1 store duy nhất là 1 object trong mô hinh tree



* Chỉ có 1 cách thay đổi state đó là tạo ra 1 action (object mô tả những gì xảy ra)
* Để chỉ rõ state tree được thay đổi bởi 1 action bạn phải viết case reducers
  1. Nguyên lý hoạt động: nhận vào 1 state, 1 action và trả về 1 newstate

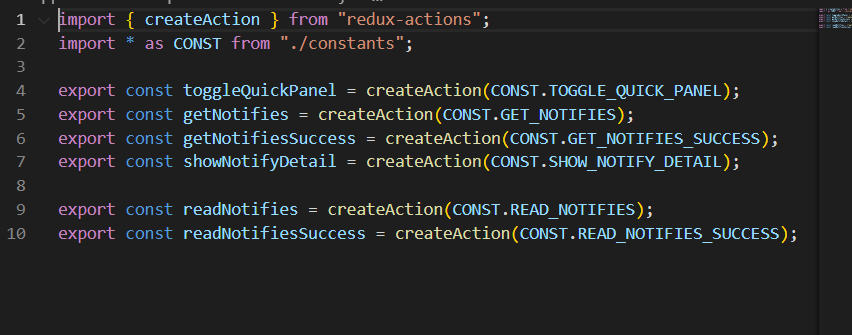


* Khởi tạo 1 Store:

import { createStore } from 'redux';

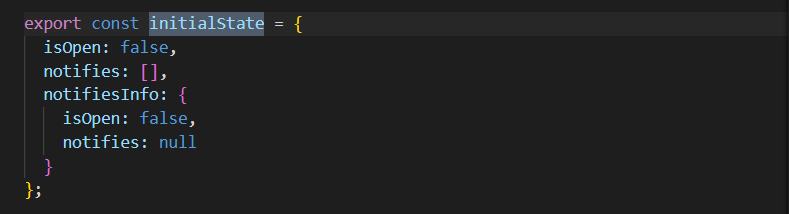
const store = createStore(myReducer);

* getState(): trả về state hiện tại
* dispatch(action): thực hiện gọi action
* subscrible(listener): lắng nghe sự thay đổi rồi lập tức cập nhật ra View
  + **Action->Reducer->Store->View**
* **Action:**  là mô tả những gì đã xảy ra và không rõ là của state nào được truyền vào store thông qua hàm dispatch()



* **Reducer:**  đảm nhiệm việc xác định hành động là của state nào và thay đổi như thế nào để trả về 1 state mới





* 1. **Smart Component (containers)**

**Containers** là những component giao tiếp với Redux thông qua **connect()**của react-redux.

**connect()** nhận vào 4 tham số **mapStateToProps mapDispatchToProps mergeProps options**:

* **mapStateToProps(state, [ownProps])**là function. Nếu được định nghĩa, container sẽ được đăng ký (subscribe) với store. Mỗi khi store update mapStateToProps sẽ được gọi, object mà nó trả về sẽ được merge với props của container. Nếu ownProps được định nghĩa, giá trị của nó sẽ là props được gửi cho container, đồng thời mỗi khi container nhận được new props thì mapStateToProps cũng sẽ được gọi. Nếu mapStateToProps không được định nghĩa container sẽ không được đăng ký và nhận update từ store.
* **mapDispatchToProps** là object hoặc function. Nếu là object mỗi function bên trong object sẽ được coi là một action creator, đồng thời tất cả function này sẽ được tự động chay bởi **bindActionCreators()** và merge chúng với props của container. Nếu là function mapDispatchToProps sẽ nhận 2 tham số (dispatch, [ownProps]), chúng ta sẽ tự định nghĩa cách bind action với dispatch, chúng ta cũng có thể sử dụng bindActionCreators({action}, dispatch) để tự động bind. Nếu ownProps được định nghĩa, giá trị của nó sẽ là props được gửi cho container, đồng thời mỗi khi container nhận được new props thì mapDispatchToProps cũng sẽ được gọi. Nếu mapDispatchToProps không được định nghĩa sẽ chỉ có dispatch được merge vào props của container.
* **mergeProps(stateProps, dispatchProps, ownProps)** là function. Nếu được định nghĩa, nó sẽ nhận vào tham số là kết qủa của mapStateToProps mapDispatchToProps và parent props. Object mà nó trả về là props được gửi cho container. Nếu không được định nghĩa **Object.assign({}, ownProps, stateProps, dispatchProps)** sẽ được sử dụng mặc định.
* **options**là object. Nếu được định nghĩa sẽ điều chỉnh hành vi của connector. Chứa 2 giá trị pure và withRef. Nếu pure = true thì thực thi **shouldComponentUpdate()**và so sánh kết qủa của mergeProps để tránh những update không cần thiết, mặc định là true. Nếu withRef = true thì lưu trữ lại ref đến container instance và có thể truy cập thông qua **getWrappedInstance()**, mặc định false.

REDUX-SAGA

Tóm gọn nội dung redux saga

Các hàm cơ bản trong redux saga:

* **Fork**(): sử dụng cơ chế non - blocking call trên function
* **Call**(): Gọi function. Nếu nó return về một promise, tạm dừng saga cho đến khi promise được giải quyết
* **Take**(): tạm dừng cho đến khi nhận được action
* **Put**(): Dùng để dispatch một action
* **takeEvery**(): Theo dõi một action nào đó thay đổi thì gọi một saga nào đó
* **akeLastest**() : Có nghĩa là nếu chúng ta thực hiện một loạt các actions, nó sẽ chỉ thực thi và trả lại kết quả của của actions cuối cùng
* **yield**(): Có nghĩa là chạy tuần tự khi nào trả ra kết quả mới thực thi tiếp
* **Select**(): Chạy một selector function để lấy data từ state

<https://viblo.asia/p/co-ban-ve-redux-saga-gDVK22x2KLj>

<https://viblo.asia/p/bat-dau-voi-redux-saga-naQZRJrmZvx>

<https://viblo.asia/p/tim-hieu-ve-react-redux-saga-xay-dung-ung-dung-don-gian-63vKjVAkK2R>

LODASH

<https://viblo.asia/p/10-lodash-functions-ma-ban-nen-biet-gDVK2RojKLj>

Sữ dụng Responsive trong React Native

<https://www.npmjs.com/package/react-native-responsive-screen>

Cách dùng Firebase cloud

1. Chức năng thêm
   1. Thêm dữ liệu bằng cách ghi đè:

db.collection("cities").doc("LA").set({  
    name: "Los Angeles",  
    state: "CA",  
    country: "USA"  
})  
.then(function() {  
    console.log("Document successfully written!");  
})  
.catch(function(error) {  
    console.error("Error writing document: ", error);  
});

* 1. Thêm dữ liệu nhưng không ghi đè mà ghi thêm những thứ chưa có:

var cityRef = db.collection('cities').doc('BJ');  
  
var setWithMerge = cityRef.set({  
    capital: true  
}, { merge: true });

* 1. Thêm dữ liệu vào Document
* Đầu tiên ta phải lấy ra được phần id trong document sau đó mới thực hiện hàm set(data).

db.collection("cities").doc("new-city-id").set(data);

* Nếu không có ID thì firebase sẽ tự động tạo cho bạn ID và bạn có thể gọi bằng phương thức add()

// Add a new document with a generated id.  
db.collection("cities").add({  
    name: "Tokyo",  
    country: "Japan"  
})  
.then(function(docRef) {  
    console.log("Document written with ID: ", docRef.id);  
})  
.catch(function(error) {  
    console.error("Error adding document: ", error);  
});

* Nếu dùng trong Document references và ID tự động thì ta có thể sữ dụng phương thức doc() như sau

// Add a new document with a generated id.  
var newCityRef = db.collection("cities").doc();  
  
// later...  
newCityRef.set(data);

1. Chức năng sửa
   1. Để cập nhật một số trường của tài liệu mà không ghi đè lên toàn bộ tài liệu, hãy sử dụng update()

var washingtonRef = db.collection("cities").doc("DC");  
  
// Set the "capital" field of the city 'DC'  
return washingtonRef.update({  
    capital: true  
})  
.then(function() {  
    console.log("Document successfully updated!");  
})  
.catch(function(error) {  
    // The document probably doesn't exist.  
    console.error("Error updating document: ", error);  
});

* 1. Nếu Document có các đối tượng lồng vào nhau ta có thế sữ dụng “” để tham chiếu tới các trường và cập nhật như sau

// Create an initial document to update.  
var frankDocRef = db.collection("users").doc("frank");  
frankDocRef.set({  
    name: "Frank",  
    favorites: { food: "Pizza", color: "Blue", subject: "recess" },  
    age: 12  
});  
  
// To update age and favorite color:  
db.collection("users").doc("frank").update({  
    "age": 13,  
    "favorites.color": "Red"  
})  
.then(function() {  
    console.log("Document successfully updated!");  
});

* 1. Khi dùng “” thì có thể thay đổi thuộc tính của từng field nhưng không ghi đè

// Create our initial doc  
db.collection("users").doc("frank").set({  
  name: "Frank",  
  favorites: {  
    food: "Pizza",  
    color: "Blue",  
    subject: "Recess"  
  },  
  age: 12  
}).then(function() {  
  console.log("Frank created");  
});  
  
// Update the doc without using dot notation.  
// Notice the map value for favorites.  
db.collection("users").doc("frank").update({  
  favorites: {  
    food: "Ice Cream"  
  }  
}).then(function() {  
  console.log("Frank food updated");  
});  
  
/\*  
Ending State, favorite.color and favorite.subject are no longer present:  
/users  
    /frank  
        {  
            name: "Frank",  
            favorites: {  
                food: "Ice Cream",  
            },  
            age: 12  
        }  
 \*/

* 1. Nếu Document có chứa các array. Ta có thể dùng phương thức arrayUnion() để có thể thêm các phần tử trong array mà chỉ thêm các phần tử chưa có. Phương thức arrayRemove() dùng để loại bỏ các yếu tố nhất định.

var washingtonRef = db.collection("cities").doc("DC");  
  
// Atomically add a new region to the "regions" array field.  
washingtonRef.update({  
    regions: firebase.firestore.FieldValue.arrayUnion("greater\_virginia")  
});  
  
// Atomically remove a region from the "regions" array field.  
washingtonRef.update({  
    regions: firebase.firestore.FieldValue.arrayRemove("east\_coast")  
});

* 1. Ta có thể tăng hoặc giảm các giá trị ở các field. Nếu field đó không tồn tại hoặc không phải giá trị số thì nó sẽ đặt lại giá trị cho field đó

var washingtonRef = db.collection('cities').doc('DC');  
  
// Atomically increment the population of the city by 50.  
washingtonRef.update({  
    population: firebase.firestore.FieldValue.increment(50)  
});

1. Chức năng load
   1. Load data từ Document

var docRef = db.collection("cities").doc("SF");  
  
docRef.get().then(function(doc) {  
    if (doc.exists) {  
        console.log("Document data:", doc.data());  
    } else {  
        // doc.data() will be undefined in this case  
        console.log("No such document!");  
    }  
}).catch(function(error) {  
    console.log("Error getting document:", error);  
});

* 1. Load data của nhiều Document từ Collection

db.collection("cities").where("capital", "==", true)  
    .get()  
    .then(function(querySnapshot) {  
        querySnapshot.forEach(function(doc) {  
            // doc.data() is never undefined for query doc snapshots  
            console.log(doc.id, " => ", doc.data());  
        });  
    })  
    .catch(function(error) {  
        console.log("Error getting documents: ", error);  
    });

* 1. Load Document từ 1 nhóm Collection
* Collection (City) có các Document

var citiesRef = db.collection('cities');  
  
var landmarks = Promise.all([  
    citiesRef.doc('SF').collection('landmarks').doc().set({  
        name: 'Golden Gate Bridge',  
        type: 'bridge'  
    }),  
    citiesRef.doc('SF').collection('landmarks').doc().set({  
        name: 'Legion of Honor',  
        type: 'museum'  
    }),  
    citiesRef.doc('LA').collection('landmarks').doc().set({  
        name: 'Griffith Park',  
        type: 'park'  
    }),  
    citiesRef.doc('LA').collection('landmarks').doc().set({  
        name: 'The Getty',  
        type: 'museum'  
    }),  
    citiesRef.doc('DC').collection('landmarks').doc().set({  
        name: 'Lincoln Memorial',  
        type: 'memorial'  
    }),  
    citiesRef.doc('DC').collection('landmarks').doc().set({  
        name: 'National Air and Space Museum',  
        type: 'museum'  
    }),  
    citiesRef.doc('TOK').collection('landmarks').doc().set({  
        name: 'Ueno Park',  
        type: 'park'  
    }),  
]);

* Group by các Document

var museums = db.collectionGroup('landmarks').where('type', '==', 'museum');  
museums.get().then(function (querySnapshot) {  
    querySnapshot.forEach(function (doc) {  
        console.log(doc.id, ' => ', doc.data());  
    });  
});

* + Để có thể thực hiện nhiều truy vấn cùng lúc thì ta nên cho onSnapshot menthod thay cho get menthod

db.collection("cities").doc("SF")  
    .onSnapshot({  
        // Listen for document metadata changes  
        includeMetadataChanges: true  
    }, function(doc) {  
        // ...  
    });