

React Native Day3

课前回顾

课堂目标

App导航框架设计

仿主流APP设计一个导航框架

欢迎页面设计

App主页设计

详情页设计

安装 react navigation 与第三方图标库 react-native-vector-icons

设计欢迎页进入主页导航

App入口引用导航

欢迎页面5秒后进入主页

设计一个转场工具类 NavigationUtil.js

欢迎页改造

主页设计底部导航

Index页面顶部导航设计

Redux 与 React Navigation结合集成

第一步：安装redux,react-redux,react-navigation-redux-helpers

第二步：配置Navigation

第二步：配置Reducer

第三步：配置store

第四步：在组件中应用

案例：使用react-navigaton+redux 修改状态栏颜色

创建Actions

创建 Actions/theme

创建Reducer/theme

在Reducer中聚合

RN网络编程

数据存储 AsyncStorage

如何使用AsyncStorage

存储数据

读取数据

删除数据

离线缓存框架设计

离线缓存有什么好处

离线缓存有什么限制

离线缓存的策略

离线缓存框架的设计

实现思想

校验时间

完整代码实例

end

## 课前回顾

- React Navigation介绍
- React Navigation概念与属性介绍
- 核心导航器的学习与使用

## 课堂目标

- 掌握react navigation 导航框架设计
- 了解redux在RN项目（使用react navigation）中的集成方式
- 掌握Fetch网络编程

## App导航框架设计

### 仿主流APP设计一个导航框架

#### 欢迎页面设计

```
import React, { Component } from "react";
import { Platform, StyleSheet, Text, View } from "react-native";
export default class WelcomePage extends Component {
  render() {
    return (
      <View style={styles.container}>
        <Text style={styles.welcome}>Welcome to WelcomePage!</Text>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#F5FCFF"
  },
  welcome: {
    fontSize: 20,
    textAlign: "center",
    margin: 10
  }
})
```

```
});
```

## App主页设计

```
import React, { Component } from "react";
import { Platform, StyleSheet, Text, View } from "react-native";

export default class HomePage extends Component {
  constructor(props) {
    super(props);
    console.disableYellowBox = true;
  }

  render() {
    return (
      <View style={styles.container}>
        <Text style={styles.welcome}>Welcome to HomePage!</Text>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#F5FCFF"
  },
  welcome: {
    fontSize: 20,
    textAlign: "center",
    margin: 10
  }
});
```

## 详情页设计

```
import React, { Component } from "react";
import { Platform, StyleSheet, Text, View } from "react-native";

export default class DetailPage extends Component {
  render() {
```

```

    return (
      <View style={styles.container}>
        <Text style={styles.welcome}>Welcome to DetailPage!</Text>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#F5FCFF"
  },
  welcome: {
    fontSize: 20,
    textAlign: "center",
    margin: 10
  }
});

```

## 安装 react navigation 与第三方图标库 react-native-vector-icons

```

yarn add react-navigation
# or with npm
# npm install --save react-navigation

yarn add react-native-gesture-handler
# or with npm
# npm install --save react-native-gesture-handler

react-native link react-native-gesture-handler

yarn add react-native-vector-icons

react-native link react-native-vector-icons

###记得关闭模拟器，服务器，重新启动项目

```

## 设计欢迎页进入主页导航

```

##### AppNavigator.js

import {
  createStackNavigator,
  createAppContainer,
  createSwitchNavigator
} from "react-navigation";

import HomePage from "../Pages/HomePage";
import WelcomePage from "../Pages/WelcomePage";
import DetailPage from "../Pages/DetailPage";

//定义欢迎导航
const AppInitNavigator = createStackNavigator({
  WelcomePage: {
    screen: WelcomePage,
    navigationOptions: {
      header: null
    }
  }
});
//定义主页导航
const AppMainNavigator = createStackNavigator({
  HomePage: {
    screen: HomePage,
    navigationOptions: {
      header: null
    }
  },
  DetailPage: {
    screen: DetailPage
  }
});

export default createAppContainer(
  createSwitchNavigator({
    Init: AppInitNavigator,
    Main: AppMainNavigator
  })
);

```

## App入口引用导航

```
import App from "../js/Navigator/AppNavigator";
```

## 欢迎页面5秒后进入主页

```
componentDidMount() {  
  this.timer = setTimeout(() => {  
    const { navigation } = this.props;  
    navigation.navigate("Main");  
  }, 1000);  
}  
componentWillUnmount() {  
  this.timer && clearTimeout(this.timer);  
}
```

## 设计一个转场工具类 NavigationUtil.js

```
export default class NavigationUtil {  
  //跳转到指定页面  
  static goPage(props, page) {  
    const navigation = NavigationUtil.navigation;  
    navigation.navigate(page, {  
      ...props  
    });  
  }  
  //go Back  
  static resetGoBack(props) {  
    const { navigation } = props;  
    navigation.goBack();  
  }  
  //回到主页  
  static resetToHomePage(params) {  
    const { navigation } = params;  
    navigation.navigate("Main");  
  }  
}
```

## 欢迎页改造

```
import NavigationUtil from "../Navigator/navigationUtil";

componentDidMount() {
  this.timer = setTimeout(() => {
    navigationUtil.resetToHomePage({
      navigation: this.props.navigation
    });
  }, 1000);
}
```

## 主页设计底部导航

```
import React, { Component } from "react";
import { Platform, StyleSheet, Text, View } from "react-native";
import {
  createAppContainer,
  createBottomTabNavigator,
} from "react-navigation";

import IndexPage from "./IndexPage";
import MyPage from "./MyPage";
import FontAwesome from "react-native-vector-icons/FontAwesome";
const TABS = {
  IndexPage: {
    screen: IndexPage,
    navigationOptions: {
      tabBarLabel: "首页",
      tabBarIcon: ({ tintColor, focused }) => (
        <FontAwesome name="home" size={26} style={{ color: tintColor }} />
      )
    }
  },
  MyPage: {
    screen: MyPage,
    navigationOptions: {
      tabBarLabel: "我的",
      tabBarIcon: ({ tintColor, focused }) => (
        <FontAwesome name="user" size={26} style={{ color: tintColor }} />
      )
    }
  }
};

export default class HomePage extends Component {
  constructor(props) {
    super(props);
    console.disableYellowBox = true;
  }
```

```

    }
    _TabNavigator() {
      return createAppContainer(createBottomTabNavigator(TABS));
    }
    render() {
      const Tabs = this._TabNavigator();
      return <Tabs />;
    }
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#F5FCFF"
  },
  welcome: {
    fontSize: 20,
    textAlign: "center",
    margin: 10
  }
});

```

## Index页面顶部导航设计

```

import React, { Component } from "react";
import { Button, Platform, StyleSheet, Text, View } from "react-native";
import {
  createAppContainer,
  createMaterialTopTabNavigator
} from "react-navigation";
import NavigationUtil from "../Navigator/NavigationUtil";

export default class IndexPage extends Component {
  constructor(props) {
    super(props);
    this.tabNames = [
      "ios",
      "android",
      "nodeJs",
      "Vue",
      "React",

```



```

    "React Native"
  ];
}
_genTabs() {
  const tabs = {};
  this.tabNames.forEach((item, index) => {
    tabs[`tab${index}`] = {
      screen: props => <IndexTab {...props} tabName={item} />,
      navigationOptions: {
        title: item
      }
    };
  });
  return tabs;
}
render() {
  const TabNavigator = createAppContainer(
    createMaterialTopTabNavigator(this._genTabs(), {
      tabBarOptions: {
        upperCaseLabel: false,
        scrollEnabled: true,
      }
    })
  );
  return (
    <View style={{ flex: 1, marginTop: 30 }}>
      <TabNavigator />
    </View>
  );
}
}

class IndexTab extends Component {
  render() {
    const { tabName } = this.props;
    return (
      <View style={styles.container}>
        <Text style={styles.welcome}>Welcome to {tabName}</Text>
        <Button
          title={"go to DetailPage"}
          onPress={() => {
            NavigationUtil.navigation.navigate("DetailPage");
          }}
        />
      </View>
    );
  }
}

```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#F5FCFF"
  },
  welcome: {
    fontSize: 20,
    textAlign: "center",
    margin: 10
  }
});
```

## Redux 与 React Navigation结合集成

Redux + React Navigation有点复杂 因为Redux是自顶向下管理一套状态，React Navigation也是自顶向下管理一套状态甚至页面，这俩融合起来就有点困难了

### 第一步：安装redux,react-redux,react-navigation-redux-helpers

```
yarn add redux
yarn add react-redux //因为redux其实是可以独立运行的js项目，但使用在react项目中，还需要使用react-redux
yarn add react-navigation-redux-helpers//在使用 React Navigation 的项目中，想要集成 redux 就必须引入 react-navigation-redux-helpers 这个库
```

### 第二步：配置Navigation

- 引入redux和react-navigation-redux-helpers

```
import { connect } from "react-redux";

import {
  createReactNavigationReduxMiddleware,
  createReduxContainer
} from "react-navigation-redux-helpers";
```

- 使用createReduxContainer方法，将RootNavigator封装成高阶组件 AppWithNavigationState，这个高阶组件完成了navigation prop的替换，改成了使用redux里的navigation

```
// 修改AppNavitor 为 RootNavigator 并不再默认导出
export const RootNavigator = createAppContainer(
  createSwitchNavigator({
    Init: AppInitNavigator,
    Main: AppMainNavigator
  })
);

const AppWithNavigationState = createReduxContainer(RootNavigator, "root");
```

- 创建导航中间件：createReduxContainer把导航状态放到props里只是能被各个组件访问到，但是React Navigation还不能识别，所以还需要最后一步——创建一个中间件，把需要导航的组件与导航reducer连接起来

```
export const middleware = createReactNavigationReduxMiddleware(
  state => state.nav,
  "root"
);
```

- 然后使用Redux的connect函数再封装一个高阶组件，默认导出

```
//State到Props的映射关系
const mapStateToProps = state => {
  return {
    state: state.nav
  };
};

//使用Redux的connect函数再封装一个高阶组件,连接 React 组件与 Redux store
export default connect(mapStateToProps)(AppWithNavigationState);
```

- 完整代码

```
import {
  createStackNavigator,
  createAppContainer,
  createSwitchNavigator
} from "react-navigation";
```

```

import HomePage from "../Pages/HomePage";
import WelcomePage from "../Pages/WelcomePage";
import DetailPage from "../Pages/DetailPage";

//引入redux
import { connect } from "react-redux";

import {
  createReactNavigationReduxMiddleware,
  // reduxifyNavigator,react-navigation-redux-helpers3.0变更,reduxifyNavigator
  被改名为createReduxContainer
  createReduxContainer
} from "react-navigation-redux-helpers";

export const rootCom = "Init"; //设置根路由

const AppInitNavigator = createStackNavigator({
  WelcomePage: {
    screen: WelcomePage,
    navigationOptions: {
      header: null
    }
  }
});

const AppMainNavigator = createStackNavigator({
  HomePage: {
    screen: HomePage,
    navigationOptions: {
      header: null
    }
  },
  DetailPage: {
    screen: DetailPage
  }
});

export const RootNavigator = createAppContainer(
  createSwitchNavigator({
    Init: AppInitNavigator,
    Main: AppMainNavigator
  })
);

/**
 * 1.初始化react-navigation与redux的中间件,
 * 该方法的一个很大的作用就是为reduxifyNavigator的key设置actionSubscribers(行为订阅者)
 */

```

```
//react-navigation-redux-helpers3.0变更,createReactNavigationReduxMiddleware的参数顺序发生了变化
export const middleware = createReactNavigationReduxMiddleware(
  state => state.nav,
  "root"
);

/* 2.将根导航器组件传递给 reduxifyNavigator 函数,
 * 并返回一个将navigation state 和 dispatch 函数作为 props的新组件;
 * 使用createReduxContainer方法,将RootNavigator封装成高阶组件
AppWithNavigationState
 * 这个高阶组件完成了navigation prop的替换,改成了使用redux里的navigation
 *
 * */

const AppWithNavigationState = createReduxContainer(RootNavigator, "root");

//State到Props的映射关系
const mapStateToProps = state => {
  return {
    state: state.nav
  };
};

//使用Redux的connect函数再封装一个高阶组件,连接 React 组件与 Redux store
export default connect(mapStateToProps)(AppWithNavigationState);
```

## 第二步：配置Reducer

```
import { combineReducers } from "redux";
import theme from "../theme";
import { rootCom, RootNavigator } from "../Navigator/AppNavigator";

//1.指定默认state
const navState = RootNavigator.router.getStateForAction(
  RootNavigator.router.getActionForPathAndParams(rootCom)
);

/**上面的代码创建了一个导航action(表示我想打开rootCom),那么我们就可以通过action创建导航
state,通过方法getStateForAction(action, oldNavigationState)
*俩参数,一个是新的action,一个是当前的导航state,返回新的状态,当没有办法执行这个action的
时候,就返回null。
**/

/**
 * 2.创建自己的 navigation reducer,
```

```

*/
const navReducer = (state = navState, action) => {
  const nextState = RootNavigator.router.getStateForAction(action, state);
  // 如果`nextState`为null或未定义, 只需返回原始`state`
  return nextState || state;
};

/**
 * 3.合并reducer
 * @type {Reducer<any> | Reducer<any, AnyAction>}
 */
const index = combineReducers({
  nav: navReducer,
  theme: theme
});

export default index;

```

### 第三步：配置store

```

import { applyMiddleware, createStore } from "redux";
import reducers from "../Reducer";
import { middleware } from "../Navigator/AppNavigator";

const middlewares = [middleware];
/**
 * 创建store
 */
export default createStore(reducers, applyMiddleware(...middlewares));

```

### 第四步：在组件中应用

```

import React, {Component} from 'react';
import {Provider} from 'react-redux';
import AppNavigator from '../Navigator/AppNavigator';
import store from './Store'

type Props = {};
export default class App extends Component<Props> {
  render() {
    /**
     * 将store传递给App框架

```

```

        */
        return <Provider store={store}>
            <AppNavigator/>
        </Provider>
    }
}

```

搞定!!

## 案例：使用react-navigaton+redux 修改状态栏颜色

### 创建Actions

```

### Types.js
export default {
    THEM_CHANGE: "THEM_CHANGE",
    THEM_INIT: "THEM_INIT"
};

```

### 创建 Actions/theme

```

import Types from "../Types";

export function onThemeChange(theme) {
    return {
        type: Types.THEM_CHANGE,
        theme: theme
    };
}

```

### 创建Reducer/theme

```

import Types from "../../Actions/Types";

const defaultState = {
    theme: "blue"
};

export default function onAction(state = defaultState, action) {
    switch (action.type) {

```

```

    case Types.THEM_CHANGE:
      return {
        ...state,
        theme: action.theme
      };
    default:
      return state;
  }
}

```

## 在Reducer中聚合

```

const index = combineReducers({
  nav: navReducer,
  theme: theme
});

```

### 1. 订阅state

```

import React, { Component } from "react";
import { Button, Platform, StyleSheet, Text, View } from "react-native";
import {
  createAppContainer,
  createMaterialTopTabNavigator
} from "react-navigation";
import IndexTab from "../Pages/IndexTab";
import { connect } from "react-redux";
import { onThemeChange } from "../Actions/theme";
import navigationUtil from "../Navigator/navigationUtil";
class IndexPage extends Component {
  constructor(props) {
    super(props);
    this.tabNames = [
      "ios",
      "android",
      "nodeJs",
      "Vue",
      "React",
      "React Native"
    ];
  }
}

```



```

_genTabs() {
  const tabs = {};
  this.tabNames.forEach((item, index) => {
    tabs[`tab${index}`] = {
      screen: props => <IndexTab {...props} tabName={item} />,
      navigationOptions: {
        title: item
      }
    };
  });
  return tabs;
}

render() {
  const TabBackground = this.props.theme;
  console.log(this.props);
  const TabNavigator = createAppContainer(
    createMaterialTopTabNavigator(this._genTabs(), {
      tabBarOptions: {
        tabStyle: {},
        upperCaseLabel: false,
        scrollEnabled: true,
        style: {
          //选项卡背景色
          backgroundColor: TabBackground
        },
        indicatorStyle: {
          //指示器的样式
          height: 2,
          backgroundColor: "#fff"
        },
        labelStyle: {
          //文字的样式
          fontSize: 16,
          marginTop: 6,
          marginBottom: 6
        }
      }
    })
  );
  return (
    <View style={{ flex: 1, marginTop: 30 }}>
      <TabNavigator />
    </View>
  );
}
}

const styles = StyleSheet.create({
  container: {

```

```

    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#F5FCFF"
  },
  welcome: {
    fontSize: 20,
    textAlign: "center",
    margin: 10
  }
});

const mapStateToProps = state => ({
  theme: state.theme.theme
});

export default connect(mapStateToProps)(IndexPage);

```

在上述代码中我们订阅了store中的theme state，然后该组件就可以通过 `this.props.theme` 获取到所订阅的theme state了。

## 2. 触发action改变state

```

import React, { Component } from "react";
import { Button, Platform, StyleSheet, Text, View } from "react-native";

import { connect } from "react-redux";
import { onThemeChange } from "../Actions/theme";

class IndexTab extends Component {
  render() {
    const { tabName } = this.props;
    return (
      <View style={styles.container}>
        <Text style={styles.welcome}>Welcome to {tabName}</Text>
        <Button
          title={"go to DetailPage"}
          onPress={() => {
            navigationUtil.goPage(this.props, "DetailPage");
          }}
        />
        <Button
          title={"改变tab背景色"}
          onPress={() => {
            this.props.onThemeChange("#000");
            // navigationUtil.goPage(this.props, "DetailPage");
          }}
        />
      </View>
    );
  }
}

```

```

    }}
  />
</View>
);
}
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#F5FCFF"
  },
  welcome: {
    fontSize: 20,
    textAlign: "center",
    margin: 10
  }
});
const mapStateToProps = state => ({});

const mapDispatchToProps = dispatch => ({
  onThemeChange: theme => dispatch(onThemeChange(theme))
});
export default connect(
  mapStateToProps,
  mapDispatchToProps
)(IndexTab);

```

## RN网络编程

React Native 提供了和 web 标准一致的[Fetch API](#)，用于满足开发者访问网络的需求。

发起请求

要从任意地址获取内容的话，只需简单地将网址作为参数传递给 fetch 方法即可（fetch 这个词本身就是 获取 的意思）

```
fetch('https://mywebsite.com/mydata.json');
```

Fetch 还有可选的第二个参数，可以用来定制 HTTP 请求一些参数。你可以指定 header 参数，或是指定使用 POST 方法，又或是提交数据等等：

```
fetch('https://mywebsite.com/endpoint/', {
  method: 'POST',
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    firstParam: 'yourValue',
    secondParam: 'yourOtherValue',
  }),
});
```

提交数据的格式关键取决于 headers 中的 `Content-Type`。`Content-Type` 有很多种，对应 body 的格式也有区别。到底应该采用什么样的 `Content-Type` 取决于服务器端，所以请和服务端开发人员沟通确定清楚。常用的 'Content-Type' 除了上面的 'application/json'，还有传统的网页表单形式，示例如下：

```
fetch('https://mywebsite.com/endpoint/', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded',
  },
  body: 'key1=value1&key2=value2',
});
```

Fetch 方法会返回一个 [Promise](#)，这种模式可以简化异步风格的代码

```
function getMoviesFromApiAsync() {
  return fetch('https://facebook.github.io/react-native/movies.json')
    .then((response) => response.json())
    .then((responseJson) => {
      return responseJson.movies;
    })
    .catch((error) => {
      console.error(error);
    });
}
```

你也可以在 React Native 应用中使用 ES2017 标准中的 `async` / `await` 语法：

```
// 注意这个方法前面有async关键字
```

```

async function getMoviesFromApi() {
  try {
    // 注意这里的await语句，其所在的函数必须有async关键字声明
    let response = await fetch(
      'https://facebook.github.io/react-native/movies.json',
    );
    let responseJson = await response.json();
    return responseJson.movies;
  } catch (error) {
    console.error(error);
  }
}

```

别忘了 catch 住 `fetch` 可能抛出的异常，否则出错时你可能看不到任何提示

注意：使用 Chrome 调试目前无法观测到 React Native 中的网络请求，你可以使用第三方的[react-native-debugger](#)来进行观测。

### 重要,处理错误

-当接收到一个代表错误的HTTP状态码时，从`fetch()`返回的promise不会被标记为reject，即使该HTTP响应的状态码是404或500。相反，它会将Promise状态标记为resolve(但是会将resolve的返回值的ok属性设置为false)，仅当网络故障时或请求被阻止时，才会被标记为reject。一次请求没有调用reject并不代表请求一定成功了，通常需要在resolve情况下，再判断`response.ok`属性为true。

```

let url = `https://api.github.com/search/repositories?q=NodeJS`;
fetch(url)
  .then(response => {
    if (response.ok) {
      return response.text();
    }
    throw new Error("Network response was not ok");
  })
  .then(responseText => {
    console.log(responseText);
  })
  .catch(e => {
    console.log(e.toString());
  });

```

## 数据存储 AsyncStorage

`AsyncStorage` 是一个简单的、异步的、持久化的 Key-Value 存储系统，它对于 App 来说是全局性的。可用来代替 `LocalStorage`。

我们推荐您在 `AsyncStorage` 的基础上做一层抽象封装，而不是直接使用 `AsyncStorage`。

在 iOS 上，`AsyncStorage` 在原生端的实现是把较小值存放在序列化的字典中，而把较大值写入单独的文件。在 Android 上，`AsyncStorage` 会尝试使用 [RocksDB](#)，或退而选择 SQLite。

## 如何使用AsyncStorage

在新版本的RN中AS已经从RN框架中移除了，使用第三方库 [react-native-community/react-native-async-storage](#) 来替代。

### 安装

```
# Install
$ yarn add @react-native-community/async-storage

# Link
$ react-native link @react-native-community/async-storage
```

### 使用

```
import AsyncStorage from '@react-native-community/async-storage';
```

### 存储数据

```
async doSave(){
  //用法1
  AsyncStorage.setItem(Key,Value,err=>{
    err && console.log(err.toString())
  })
  //用法2
  AsyncStorage.setItem(Key,Value)
  .catch(e=>{
    err && console.log(err.toString())
  })
  //用法3
  try{
    await AsyncStorage.setItem(Key,value)
  }catch(err){
    err && console.log(err.toString())
  }
}
```

## 读取数据

```
async getData(){
  //用法1
  AsyncStorage.getItem(Key, (err, value)=>{
    console.log(value)
    err && console.log(err.toString())
  })
  //用法2
  AsyncStorage.getItem(Key)
  .then(value=>{
    console.log(value)
  })
  .catch(e=>{
    err && console.log(err.toString())
  })
  //用法3
  try{
    const value = await AsyncStorage.getItem(Key)
    console.log(value)
  }catch(err){
    err && console.log(err.toString())
  }
}
```

## 删除数据

```
async doRemove(){
  //用法1
  AsyncStorage.removeItem(Key, (err)=>{
    err && console.log(err.toString())
  })
  //用法2
  AsyncStorage.removeItem(Key)
  .catch(e=>{
    err && console.log(err.toString())
  })
  //用法3
  try{
    await AsyncStorage.removeItem(Key)
  }catch(err){
    err && console.log(err.toString())
  }
}
```

# 离线缓存框架设计

## 离线缓存有什么好处

- 提升用户体验，用户的网络情况我们不能控制，但是我们可以离线存储提升体验。
- 节省流量：节省服务器流量，节省用户手机的流量

## 离线缓存有什么限制

数据的实时性要求不高，推荐使用

## 离线缓存的策略

- 优先从本地获取数据，如果数据过时或者不存在，则从服务器获取数据，数据返回后同时将数据同步到本地数据库。
- 优先从服务器获取数据，数据返回后同步到本地数据库，如果发生网络故障，才从本地获取数据。

## 离线缓存框架的设计

按照第一个策略：如果数据过时或者不存在，则从服务器获取数据，数据返回后同时将数据同步到本地数据库。

- 优先从本地获取数据
- 如果数据存在且在有效期内，我们将数据返回
- 否则获取网络数据

## 实现思想

```
fetchData(url){
  return new Promise((resolve,reject)=>{
    //获取本地数据
    this.fetchLocalData(url)
      .then((wrapdata)=>{
        //检查有效期
        if(wrapdata && DataStore.checkTimestampValid(wrapdata.timestamp)){
          resolve(wrapdata)
        }else{
          //获取网络数据
          this.fetchNetData(url)
            .then((data)=>{
              //给数据打个时间戳
              resolve(this._wrapData(data))
            })
        }
      })
  })
}
```



```

        })
        .catch((e) => {
            reject(e)
        })
    }
})
.catch(error => {
    this.fetchNetData(url)
    .then(data => {
        resolve(this._wrapData(data));
    })
    .catch(error => {
        reject(error);
    });
});
})
}

```

首先需要实现对数据的存储:

```

import AsyncStorage from "@react-native-community/async-storage";

export default class DataStore {
    saveData(url, data, callback) {
        if (!data || !url) return;
        AsyncStorage.setItem(url, JSON.stringify(this._wrapData(data)), callback);
    }
}

```

//上述代码 url作为缓存数据的key,接受一个Object的参数data为value,因为AS是无法保存object的,所以需要把它序列化成json

给离线的数据添加一个时间戳, 便于计算有效期

```

_wrapData(data) {
    return {data: data, timestamp: new Date().getTime()};////本地时间, 推荐服务器时间
}

```

获取本地数据

```

fetchLocalData(url){
  return new Promise((resolve,reject)=>{
    AsyncStorage.getItem(url,(err,result)=>{
      if(!err){
        resolve(JSON.parse(result))// getItem获取到的是string, 我们需要将其反序列化为object
      }else{
        reject(err);
        console.log(err)
      }
    })
  })
}

```

获取网络数据

```

fetchNetData(url){
  return new Promise((resolve,reject)=>{
    fetch(url)
      .then((response)=>{
        if(response.ok){
          return response.json();
        }
        throw new Error('network response was not ok')
      })
      .then((responseData)=>{
        this.saveData(url,responseData);
        resolve(responseData);
      })
      .catch((e)=>{
        reject(e)
      })
  })
}

```

校验时间

```

static checkTimestampValid(timestamp) {
  const currentDate = new Date();
  const targetDate = new Date();
  targetDate.setTime(timestamp);
  if (currentDate.getMonth() !== targetDate.getMonth()) return false;
  if (currentDate.getDate() !== targetDate.getDate()) return false;
  if (currentDate.getHours() - targetDate.getHours() > 4) return false; //有效期
4个小时
  // if (currentDate.getMinutes() - targetDate.getMinutes() > 1) return false;
  return true;
}

```

## 完整代码实例

```

import AsyncStorage from "@react-native-community/async-storage";

export default class DataStore {

  static checkTimestampValid(timestamp) {
    const currentDate = new Date();
    const targetDate = new Date();
    targetDate.setTime(timestamp);
    if (currentDate.getMonth() !== targetDate.getMonth()) return false;
    if (currentDate.getDate() !== targetDate.getDate()) return false;
    if (currentDate.getHours() - targetDate.getHours() > 4) return false; //有
效期4个小时
    // if (currentDate.getMinutes() - targetDate.getMinutes() > 1) return
false;
    return true;
  }

  fetchData(url) {
    return new Promise((resolve, reject) => {
      //获取本地数据
      this.fetchLocalData(url)
        .then(wrapdata => {
          //检查有效期
          if (wrapdata && DataStore.checkTimestampValid(wrapdata.timestamp)) {
            resolve(wrapdata);
          } else {
            //获取网络数据
            this.fetchNetData(url)

```

```

        .then(data => {
            //给数据打个时间戳
            resolve(this._wrapData(data));
        })
        .catch(e => {
            reject(e);
        });
    }
})
.catch(error => {
    this.fetchNetData(url)
        .then(data => {
            resolve(this._wrapData(data));
        })
        .catch(error => {
            reject(error);
        });
});
});
}

saveData(url, data, callback) {
    if (!data || !url) return;
    AsyncStorage.setItem(url, JSON.stringify(this._wrapData(data)), callback);
}

_wrapData(data) {
    return { data: data, timestamp: new Date().getTime() }; //本地时间, 推荐服务器
    时间
}

fetchLocalData(url) {
    return new Promise((resolve, reject) => {
        AsyncStorage.getItem(url, (err, result) => {
            if (!err) {
                resolve(JSON.parse(result)); // getItem获取到的是string, 我们需要将其反序
                列化为object
            } else {
                reject(err);
                console.log(err);
            }
        });
    });
}

fetchNetData(url) {
    return new Promise((resolve, reject) => {
        fetch(url)
            .then(response => {

```

```

        if (response.ok) {
            return response.json();
        }
        throw new Error("network response was not ok");
    })
    .then(responseData => {
        this.saveData(url, responseData);
        resolve(responseData);
    })
    .catch(e => {
        reject(e);
    });
    });
}
}

```

测试:

```

import React, { Component } from "react";
import { Button, Platform, StyleSheet, Text, View } from "react-native";

import DataStore from "../Http/AsDemo";

class TestItem extends Component {
  constructor(props) {
    super(props);
    this.dataStore = new DataStore();
  }
  componentDidMount() {
    let url = `https://api.github.com/search/repositories?q=NodeJS`;
    this.dataStore
      .fetchData(url)
      .then(response => {
        console.log(response);
      })
      .catch(e => {
        console.log(e);
      });
  }
  render() {
    return (
      <View style={styles.container}>
        <Text>测试缓存</Text>
      </View>
    );
  }
}

```

```
}  
}  
  
const styles = StyleSheet.create({  
  container: {  
    flex: 1,  
    justifyContent: "center",  
    alignItems: "center",  
    backgroundColor: "#F5FCFF"  
  },  
  welcome: {  
    fontSize: 20,  
    textAlign: "center",  
    margin: 10  
  },  
  instructions: {  
    textAlign: "center",  
    color: "#333333",  
    marginBottom: 5  
  }  
});
```

**end**