Template

Panel Development > Panel SDK Development > 5 Min > Template

Version: 20200217

Online Version

Contents

Tuya Panel Kit Template is the best practice of device control panel engineering, including the basic framework of the development panel, this document will introduce What templates do for us, and how we need to develop a brand new panel project based on templates.

1 Run start

Please refer to the previous quick start article

2 Detailed template

2.1 Detailed directory

```
1     .babelrc              // babel configuration file
2     .eslintignore         // Configure which files don't need eslint
3     .eslintrc.js          // eslint configuration file
4     .gitignore            // Configure which files don't require git
5     .npmrc                // npm configuration file
6     README.md             // Project information, including but not
      limited to project name, productId, author, description, etc.
7     index.android.js      // Android entry
8     index.ios.js          // IOS entry
9     index.js              // Android portal (for compatibility)
10    package.json          // npm dependency management
11    rn-cli.config.js      // metro configuration file
12    src
13        components        // Place each reused functional component used
       in the project
14        composeLayout.js // The package handles some `device events`
      and` device information` needed inside the panel
15        containers        // Place each page-level component of the
      project
16        i18n              // Place Multilingual Profile
17        main.js           // The project entry file, inherited from `
      NavigatorLayout`, passes some necessary configuration, such as
      background, topbar, etc., by overriding the` hookRoute` method.
      Override the `renderScene` method to control route forwarding.
18        redux             // Place some code related to redux
19        res               // Place local resources, including pictures,
      svg path, etc.
20        utils             // Some common functions that will be used
      inside the panel
21    yarn.lock
```

2.2 Detailed composeLayout

In simple terms, there are three things that the higher-order function composeLayout does for us;

1. When the panel is initialized, `composeLayout` processes the original`devInfo` and initializes the `redux store`.
2. When the panel is running, `composeLayout` listens to events related to device changes and updates the corresponding`redux store` in real time.

3. The incoming component is connected to the redux store.

```
import _ from "lodash";
import PropTypes from "prop-types";
import React, { Component } from "react";
import { Provider, connect } from "react-redux";
import { TYSdk, Theme } from "tuya-panel-kit";
import {
  devInfoChange,
  deviceChange,
  responseUpdateDp
} from "./redux/modules/common";
import theme from "./config/theme";

const TYEvent = TYSdk.event;
const TYDevice = TYSdk.device;

/**
 *
 * @param {Object} store - redux store
 * @param {ReactComponent} component - The component that needs to be
     connected to the redux store, which is usually main
 */
const composeLayout = (store, component) => {
  const NavigatorLayoutContainer = connect(_.identity)(component);
  const { dispatch } = store;

  /**
   * The `device data change` event is monitored here,
   * Whenever the dp point data is changed, the changed dp point status
       will be updated to `redux` synchronously.
   * Similarly, when the device information is changed, the changed
       device information value will also be updated to `redux`
       synchronously.
   */
  TYEvent.on("deviceDataChange", data => {
    switch (data.type) {
      case "dpData":
        dispatch(responseUpdateDp(data.payload));
        break;
      default:
        dispatch(deviceChange(data.payload));
        break;
    }
  });

  /**
   * The `Network State Change Event` event is monitored here,
   * Whenever the device information is changed, the changed device
       information value will be updated to `redux` synchronously.
   */
  TYEvent.on("networkStateChange", data => {
    dispatch(deviceChange(data));
  });

  class PanelComponent extends Component {
    static propTypes = {
      // eslint-disable-next-line
      devInfo: PropTypes.object.isRequired
    };

    /**
```

2.3 Detailed main

Simply put, there is only one thing that the MainLayout entry component does for us;

1. Inherited NavigatorLayout to help the panel manage multiple pages internally.

```
import _ from "lodash";
import React from "react";
import { TYSdk, NavigatorLayout } from "tuya-panel-kit";
import composeLayout from "./composeLayout";
import configureStore from "./redux/configureStore";
import Home from "./containers/Home";
import { formatUiConfig } from "./utils";

console.disableYellowBox = true;

/**
 * Generate a store through the built-in store configuration method of the template.
 */
export const store = configureStore();

class MainLayout extends NavigatorLayout {
  constructor(props) {
    super(props);
    console.log("TYSdk :", TYSdk);
  }

  /**
   *
   * @desc
   * hookRoute can do some control processing for specific routes here
   *
   * @param {Object} route
   * @return {Object} - Some control values provided to the parent container layout of the current page component
   * {
   * style: ViewPropTypes.style, // container style, you can adjust the background color here
   * background: backgroundImage | linearGradientBackground, // panel image background or gradient background, the gradient format can refer to LinearGradient and RadialGradient components
   * topbarStyle: ViewPropTypes.style, // TopBar style, can adjust TopBar background color
   * topbarTextStyle: Text.propTypes.style, // TopBar text style
   * renderTopBar: () => (), // custom render TopBar
   * hideTopbar: true | false, // control whether to hide TopBar
   * renderStatusBar: () => {}, // custom render StatusBar, IOS only
   * showOfflineView: true | false, // control whether to render OfflineView
   * OfflineView: ReactComponent, // custom OfflineView component
   * }
   */
  // eslint-disable-next-line
  hookRoute(route) {
    // switch (route.id) {
    //   case 'main':
    //     // eslint-disable-next-line
    //     route.background = background;
    //     break;

    //   default:
    //     break;
    // }

    return {};
```

2.4 devInfo Detailed

Earlier in NavigatorLayout documentation, the unprocessed `devInfo` object has been explained, and here The `devInfo` object processed by composeLayout is also a common and important thing inside the panel. For the subsequent use of devInfo inside the panel, please refer to the following. The following will explain the role of some common fields.

- name: Device name
- productId: product id
- uiId: panel id corresponding to the current product
- bv: hardware baseline version
- devId: device Id
- gwId: Gateway Id, if it is a single product, devId is generally equal to gwId
- ability: Only for Bluetooth devices, if it is a single-point Bluetooth device, the value is 5
- AppOnline: App is online
- deviceOnline: Whether the device is online
- isLocalOnline: Whether the LAN is online
- isShare: Whether it is a shared device
- isVDevice: whether it is a demo device
- groupId: group device Id, can be used to determine whether the group device
- networkType: The online type of the device
- capability: the capability type of the device, indicating what capabilities the device supports, such as ZigBee, infrared, Bluetooth, etc.
- schema: Definition of function point (dp, data point) of the product to which the device belongs. For the explanation of function points, please refer to dp explanation
- state: state of the dp point