1. 准备工作

我们将会使用到以下工具：

- Node.js (www.nodejs.org)

  Node.js是一个Javascript运行环境，它封装了Google V8引擎。可以用在服务器环境中，也可以作为命令行（CLI）工具的运行环境。Node.js有两个发行分支，一个是长期支持（LTS）版本，一个为最新版本。生产和测试环境中可以使用LTS版本保证稳定性，开发时可使用最新版本。

- npm

  其全称是Node Package Manager，是一个NodeJS包管理和分发工具。已经集成在了Node.js的安装包中。

- Yeoman (www.yeoman.io)

  Yeoman将其定义为一个JavaScript项目“脚手架”（Scaffolding）工具。可以通过其命令行工具（yo）来完成很多重复操作。它提供了非常多的模板，用来生成不同类型的 Web 应用。这些模板称为生成器（generator）。可以通过npm或者yo来安装新的生成器。

- Cmder (cmder.net) （可选，用于Windows）

  Cmder是Windows 平台下一款具有良好用户体验的命令行工具，支持多Tab显示。

其他使用的工具通常会由yeoman生成器来自动安装，并以npm script方式来调用，不需要手动安装，因此将不再详述。

1.1 安装Node.js

首先，从Node.js官方网站下载其最新版本进行安装。安装完成后，在命名行中执行以下命令以查看Node和npm版本。

```
$ node -v
v6.1.0
$ npm -v
3.8.6
```

1.2 安装Yeoman及其生成器

Yeoman本身作为一个npm包，可以直接通过npm进行安装：

```
$ npm install -g yo
```

-g表示全局安装，因为我们是将yo作为一个全局命令行工具来进行安装的，所以一定记得添加此选项。

安装完成后，执行以下命令：

```
$ yo
  Run a generator
  ---------------
> Install a generator
  Find some help
  Get me out of here!
```

用键盘上下键可选择菜单，其中最后一项为退出yo。在没有安装任何生成器时，默认菜单项为`Install a generator`，按回车执行此菜单。

此时yo提示如下：

```
? Search npm for generators: _
```

输入关键词后，yo将搜索npm中所有以`generator-`开头，并包括输入的关键词的包。为了搭建React+Redux应用，并使用webpack作为打包器，我们可以输入`react-webpack-redux`，并回车。

在返回结果中选择`react-webpack-redux`，安装此生成器。安装完成后退出yo。

我们也可以直接使用npm安装此生成器，如下所示：

```
$ npm install -g generator-react-webpack-redux
```

详细信息可查看此生成器的[Github主页 (https://github.com/stylesuxx/generator-react-webpack-redux)](https://github.com/stylesuxx/generator-react-webpack-redux)，其说明文档中包括了一些常用命令说明。

2. 创建项目

首先建立项目文件夹并设为当前工作目录：

```
$ mkdir react-tutorial && cd react-tutorial
```

使用yo生成项目框架：

```
$ yo react-webpack-redux
```

按照提示输入项目的信息，包括：

1. 项目应用名称
2. 样式语言类型，此生成器支持多种CSS预处理器，如不使用，可选择默认值css
3. 是否启用postcss库，默认为不启用

然后此生成器将创建项目框架，生成必要的目录和文件，并安装工具依赖的npm包。其中主要的目录和文件有：

- *dist*: 发布文件目录。webpack将打包会的文件放置在此目录中
- *node_modules*: Node local模块安装目录
- *src*: 应用源代码目录。其中各个子目录对应React及Redux的某类组件（component/container/reducer等）
- *test*: 测试代码目录
- *.babelrc*: BabelJS配置文件。BabelJS用于将JSX和ES6代码转译为ES5代码
- *.eslintrc*: ESLint配置文件。ESLint为JavaScript代码静态检查工具
- *.gitignore*: Git忽略文件列表。使用SVN时，需要将其中的项目加入SVN忽略文件列表，尤其是node_modules目录，否则将导致提交大量不必要的文件到SVN中。

- *.yo-rc.json*: yo项目配置文件
- *karma.conf.js*: karma配置文件。Karma是JavaScript测试工具
- *package.json*: npm包定义文件，包含包（应用程序）信息、依赖项、开发依赖项、脚本等
- *server.js*: webpack开发服务器启动主文件
- *webpack.config.js*: webpack配置文件

yo 生成器退出后，执行以下命令以运行项目：

```
$ npm start
```

npm将会自动打开一个浏览器页面，我们将会看到以下信息输出：

```
Warning while compilling. App ready.
```

这表明应用已经正常运行，只是在编译时发生了一些警告。我们可以在命令行窗口，以及浏览器开发者工具的Console页签中看到详细的警告信息。

3. pacakge.json

首先我们用任意文本编辑器（推荐Atom）打开项目目录下的package.json文件。

可以在其scripts对象的start属性中看到以下脚本：

```
{
  "scripts": {
    "start": "node server.js --env=dev",
  }
}
```

也就是我们在执行npm start时，实际会执行node server.js --env=dev脚本，启动开发环境下的webpack开发服务器。而此服务器是支持自动热加载的，对项目的修改并不需要重新启动项目，甚至不需要我们手动刷新浏览器，这为开发提供了极大的便利，提升了工作效率。

然后在package.json中可以看到以下依赖项：

```
{
  "devDependencies": {
    "babel-core": "^6.0.0",
    "babel-eslint": "^6.0.0"
  },
  "dependencies": {
    "core-js": "^2.0.0",
    "normalize.css": "^4.0.0",
    "react": "^15.0.0",
    "react-dom": "^15.0.0",
    "react-redux": "^4.4.5",
    "redux": "^3.5.2"
  }
}
```

devDependencies中定义的是开发依赖项，即在开发过程中作用的打包、测试、代码检查等所需的依赖包，而dependencies中定义的是项目运行时所需的依赖项，需要最终打包进发布版本中。

4. 编码管理开发

4.1 添加编码管理容器

React中的容器（container)负责状态数据管理，我们计划将编码管理的功能放在一个名为Coding的container中。首先通过生成器创建此container:

```
$ yo react-webpack-redux:container Coding
```

生成器创建了一个文件*src/containers/Coding.js*，包含了一些模板代码。为了在首页中看到此container，我们对*src/index.js*文件修改为如下：

```
import React from 'react';
import { render } from 'react-dom';
import { Provider } from 'react-redux';
import configureStore from './stores';
import Coding from './containers/Coding';

const store = configureStore();

render(
  <Provider store={store}>
    <Coding />
  </Provider>,
  document.getElementById('app')
);
```

然后将*src/containers/Coding.js*的render()修改为：

```
render() {
  const {actions} = this.props;
  return (
    <div>
      <h3>Hello Coding</h3>
    <div>
  );
}
```

保存，然后返回浏览器窗口，你将会看到黑底白字的“Hello Coding"。

4.2 添加编码分类树型组件

我们将编码管理的组件都放在coding目录下，首先通过生成器创建此组件：

```
$ yo react-webpack-redux:component coding/Tree
```

生成器创建的文件位于*src/component/coding/TreeComponent.js*，同时还会生成相关的css文件（*src/styles/coding/Tree.cs*）和单元测试文件（*test/component/coding/TreeComponentTest.js*）。

修改Coding容器以显示此组件：

```
import CodingTree from '../components/coding/TreeComponent';

class Coding extends Component {
  render() {
    const {actions} = this.props;
    return (
      <div>
        <CodingTree/>
      </div>
      );
  }
}
```

保存，然后返回浏览器窗口，可以看到新的组件已经显示。

接下来，我们使用Material-UI库中的嵌套列表来实现树型显示。首先使用npm来安装Material-UI：

```
$ npm install material-ui --save
```

我们并没用使用-g选项，因为Material-UI是作为项目的本地（local）依赖来使用的，只要安装到项目目录的*node_modules*目录中即可。而--save选项则是将其保存到*package.json*的dependencies部分中，这样在将*package.json*提交到版本管理中去后，其他同事只要运行npm install即可安装所有的依赖项。--save-dev选项则是将包添加到*package.json*的devDependencies部分中。

material-ui安装完成后，我们注意到有一条提醒信息：

> material-ui@0.15.0 requires a peer of react-tap-event-plugin@^1.0.0 but none was installed.

这说明material-ui有一个依赖项没有安装。通常每个包会定义自己的依赖项，在通过npm安装时会自动安装依赖，但少数特殊的依赖则需要用户根根据自己的需求来安装。我们接下来安装此依赖项：

```
$ npm install react-tap-event-plugin@^1.0.0
```

@^1.0.0是一个版本指示，表明安装主版本号为1的最新版本。npm采用的是一种语义版本号（Semantic versioning）。详见node-semver [Github主页 (https://github.com/npm/node-semver)](https://github.com/npm/node-semver)。

然后如下修改*TreeComponent.js*, 添加Material-UI中的子组件：

```
import React from 'react';

import {List, ListItem} from 'material-ui/List';
import ActionGrade from 'material-ui/svg-icons/action/grade';
import ContentInbox from 'material-ui/svg-icons/content/inbox';
import ContentDrafts from 'material-ui/svg-icons/content/drafts';
import ContentSend from 'material-ui/svg-icons/content/send';
import Subheader from 'material-ui/Subheader';

require('styles/coding/Tree.css');

class TreeComponent extends React.Component {
  render() {
    return (
      <List className="tree-component">
        <Subheader>Nested List Items</Subheader>
        <ListItem primaryText="Sent mail" leftIcon={<ContentSend />} />
        <ListItem primaryText="Drafts" leftIcon={<ContentDrafts />} />
        <ListItem
          primaryText="Inbox"
          leftIcon={<ContentInbox />}
          initiallyOpen={true}
          primaryTogglesNestedList={true}
          nestedItems={[
            <ListItem
              key={1}
              primaryText="Starred"
              leftIcon={<ActionGrade />}
            />,
            <ListItem
              key={2}
              primaryText="Sent Mail"
              leftIcon={<ContentSend />}
              disabled={true}
              nestedItems={[
                <ListItem key={1} primaryText="Drafts" leftIcon={<ContentDrafts
/>} />
              ]}
            />
          ]}
        />
      </List>
    );
  }
}
```

同时修改Coding容器，以添加Material-UI主题支持：

```
import getMuiTheme from 'material-ui/styles/getMuiTheme';
import MuiThemeProvider from 'material-ui/styles/MuiThemeProvider';
import CodingTree from '../components/coding/TreeComponent';

class Coding extends Component {
  render() {
    const {actions} = this.props;
    return (
      <MuiThemeProvider muiTheme={getMuiTheme()}>
        <CodingTree/>
      </MuiThemeProvider>
      );
  }
}
```

然后就可以在浏览器中看到以下的树形显示：