3D巡检项目前端学习文档

1. 主要使用前端技术及版本说明

prettier 代码规范管理

antd ^4.8.2 ui框架

axios ^0.24.0

react 17.x

redux ^6.0.0

soonspacejs ^2.3.2

video.js ^7.15.4

typescript4

1. 数据流方案 ———— Dva

Dva初步介绍：

dva 首先是一个基于redux和redux-saga的数据流方案，然后为了简化开发体验，dva 还额外内置了react-router和fetch，所以也可以理解为一个轻量级的应用框架。

Dva解决的问题

* 文件切换问题。redux 的项目通常要分 reducer, action, saga, component 等等，他们的分目录存放造成的文件切换成本较大。
* 不便于组织业务模型 (或者叫 domain model) 。比如我们写了一个 userlist 之后，要写一个 productlist，需要复制很多文件。
* saga 创建麻烦，每监听一个 action 都需要走 fork -> watcher -> worker 的流程
* entry 创建麻烦。可以看下这个redux entry的例子，除了 redux store 的创建，中间件的配置，路由的初始化，Provider 的 store 的绑定，saga 的初始化，还要处理 reducer, component, saga 的 HMR 。这就是真实的项目应用 redux 的例子，看起来比较复杂。

Dva 核心概念

**基于 Redux 理念的数据流向。** 用户的交互或浏览器行为通过 dispatch 发起一个 action，如果是同步行为会直接通过 Reducers 改变 State，如果是异步行为（可以称为副作用）会先触发 Effects 然后流向 Reducers 最终改变 State。



**基于 Redux 的基本概念。**包括：

State 数据，通常为一个 JavaScript 对象，操作的时候每次都要当作不可变数据（immutable data）来对待，保证每次都是全新对象，没有引用关系，这样才能保证 State 的独立性，便于测试和追踪变化。

Action 行为，一个普通 JavaScript 对象，它是改变 State 的唯一途径。

dispatch，一个用于触发 action 改变 State 的函数。

Reducer 描述如何改变数据的纯函数，接受两个参数：已有结果和 action 传入的数据，通过运算得到新的 state。

Effects（Side Effects） 副作用，常见的表现为异步操作。dva 为了控制副作用的操作，底层引入了redux-sagas做异步流程控制，由于采用了generator 的相关概念，所以将异步转成同步写法，从而将 effects 转为纯函数。

Connect 一个函数，绑定 State 到 View代码结构目录

**其他概念**

Subscription，订阅，从源头获取数据，然后根据条件 dispatch 需要的 action，概念来源于elm。数据源可以是当前的时间、服务器的 websocket 连接、keyboard 输入、geolocation 变化、history 路由变化等等。

Router，前端路由，dva 实例提供了 router 方法来控制路由，使用的是react-router。

Route Components，跟数据逻辑无关的组件。通常需要 connect Model 的组件都是 Route Components，组织在/routes/目录下，而/components/目录下则是纯组件（Presentational Components，详见组件设计方法）

1. soonspace常用函数相关
2. **loadSbm（）**函数

加载sbm模型

interface SbmInfo extends BaseObject3DInfo, ObjectEvents<Sbm> {

 url: string;

}

function loadSbm(

 sbmInfo: SbmInfo,

 onProgress?: ModelLoadingProgressCallback

): Promise<Sbm>;

参数 sbmInfo 其中url和id为必填

1. **createPoi（）函数**

创建poi对象

interface PoiInfo extends BaseObject3DInfo, ObjectEvents<Poi> {

 url: string;

 nameScale?: Scale;

}

function createPoi(poiInfo: PoiInfo): Poi;

1. getSbmById（）函数

通过id查找sbm对象

1. unEdgeShow（）函数

取消模型轮廓，如没有传参, 将取消整个场景内所有模型的轮廓

1. edgeShow（）函数

与上面相反

1. clearTopology（）函数

清除当前场景内所有 topology 对象。

1. clearCanvas3D（）函数

清除当前场景内所有 Canvas3D 对象

1. getCameraViewpoint（）函数

获取当前相机视角数据

Position ————相机位置

Rotation————相机旋转弧度

1. flyToObj（）函数

相机飞向对象，传入值

1. flyTo（）函数

相机飞向固定位置

传入值position rotation

1. clearPoiNode（）

清除当前场景内所有 poiNode 对象

1. opacityShow（）

透明显示模型（）

1. registerPlugin（）

注册插件（）

1. plugin-patrol-controls

路径巡检控制器

Start（） 开始巡检

定义：

interface StartOptions {

 eyeHeight?: number

 naviSpeed?: number

 rotateSpeed?: number

 flyToStartPoint?: boolean

 onUpdate?: (e: Position) => {}

 onEnd?: (endPosition: Position) => {}

}

function start(path: Topology, options: StartOptions) => void

用法：patrolControls.start(

 // path

 shortestPath.

 // options

 {

 naviSpeed: 1,

 rotateSpeed: 1,

 eyeHeight: 100,

 flyToStartPoint: true,

 onUpdate: (realTimePosition) => {

 console.log(realTimePosition)

 },

 onEnd: (position) => {

 console.log('巡检结束！', position)

 }

 }

)

参数

Path

描述: 用于自动巡检的路径

必填:

类型: Topology

Options

不是必填

naviSpeed 巡检时导航速度 number 1

rotateSpeed 视角旋转速度 number 1

eyeHeight 眼睛高度 number 100

flyToStartPoint 是否飞向起始点位置 boolean true

**onUpdate 巡检时实时更新回调函数 (realTimePosition: Position) => void**

onUpdate 巡检结束回调函数 (position: Position) => void

stop（）结束巡检

Pause（）暂停巡检

Resume（）继续巡检

**项目入口文件src\pages\Home\Firefighting\index.jsx**

文件中包含两个函数Firefighting和mapStateToProps

最后使用了connect(mapStateToProps)(Firefighting)将数据mapStateToProps和Firefighting

Connect函数从umi（集成dva）中导出

**数据相关（获取基本上整个项目所有设备相关数据并根据所处位置分类）**

1. 调用/api/pl-files接口获取文件资源根路径，将BaseUrl通过dispatch至全局变量中
2. 调用/api//model/allModelCategory接口将获取的所有设备类保存至sessionStorage中的allModelCategory和allModelSystem中
3. 调用/api/pl-model-all-child-instanceds接口获取所有区域的3D模型位置处理后保存至sessionStorage，全局变量更新
4. 根据上面获取的设备类和具体设备进行分组分类

**界面相关**

**<Loading />组件根据ssp的加载情况显示加载动画，加载完成后动画小时**

响应式数据const [sceneData, setSceneData] = useState(null);

**<SoonspaceScene/>**组件需要传入参数sceneData来自pl-model-all-child-instanceds接口

组件渲染完成返回ssp参数

根据ssp是否为空参数来渲染左上显示区域切换面板，左下可操作设备信息，右侧巡检计划及记录列表，底部分类信息查看

**三维渲染组src\component\SoonspaceScene2\index.jsx**

将传入的数据modelType区分为3D和2D数据 使用format函数将设备信息转换为对象并绑定事件，获取url，用soonpace中的ssp中的load函数加载三维图形，加载视频组件，执行加载完模块回调函数

**场景控制组件ThumbnailComponent**

本组件需要了解的函数：

onShowCurrentModel函数-----修改三维视图上面的显示的Model

入参number | string string

为‘all’时将所有model全部显示state置0，否则修改相应楼层的state为0。

handleModelShrinkSpread（bool: boolean）函数

修改store中isShrink变量来表示是否拆分

modelOpcity（）变更透明度

组件功能

复位视图按钮 调用onShowCurrentModel('all')函数将站厅站台设备层显示，调用cameraView（）函数使用soonspace对象ssp中的flyTo函数将摄像头移动到初始位置

全部楼层点击调用onShowCurrentModel('all')函数将站厅站台设备层显示

拆分整体点击调用handleModelShrinkSpread（）

长方体点击事件 onShowCurrentModel(model.id)将点击的模块相应展示其余隐藏

显示透明隐藏按钮调用onSelectModel（）修改函数状态

使用useEffect监测modelShowId的变化调用modelOpcity（）

**画面可操作设备信息OperateDeviceList**

onOperateDeviceItemClick (type, item)函数

type:label时

调用clearPoiNode 清除当前场景内所有 poiNode 对象

unEdgeShow取消模型轮廓替换selectDeviceDetail变量的数据

selectDeviceDetail使用useEffect

根据新数据调用api/model/modelInformation/hierarchy接口查询poi所需的数据

createPoiNode创建poi点

type: total时

调用flyToObj

设备列表

Select筛选框 获取sessionStorage中allModelSystem遍历，存储点击的id

下方只显示被选中分类的

调用onOperateDeviceItemClick更新三维界面展示的数据

**设备巡检MaintenancePanel**

根据选中的已建计划，记录渲染不同组件



**计划组件Survey**



**记录组件**

****

**记录组件的弹框 ReportModal**

****

**EditPlan组件**



**执行已建计划**

