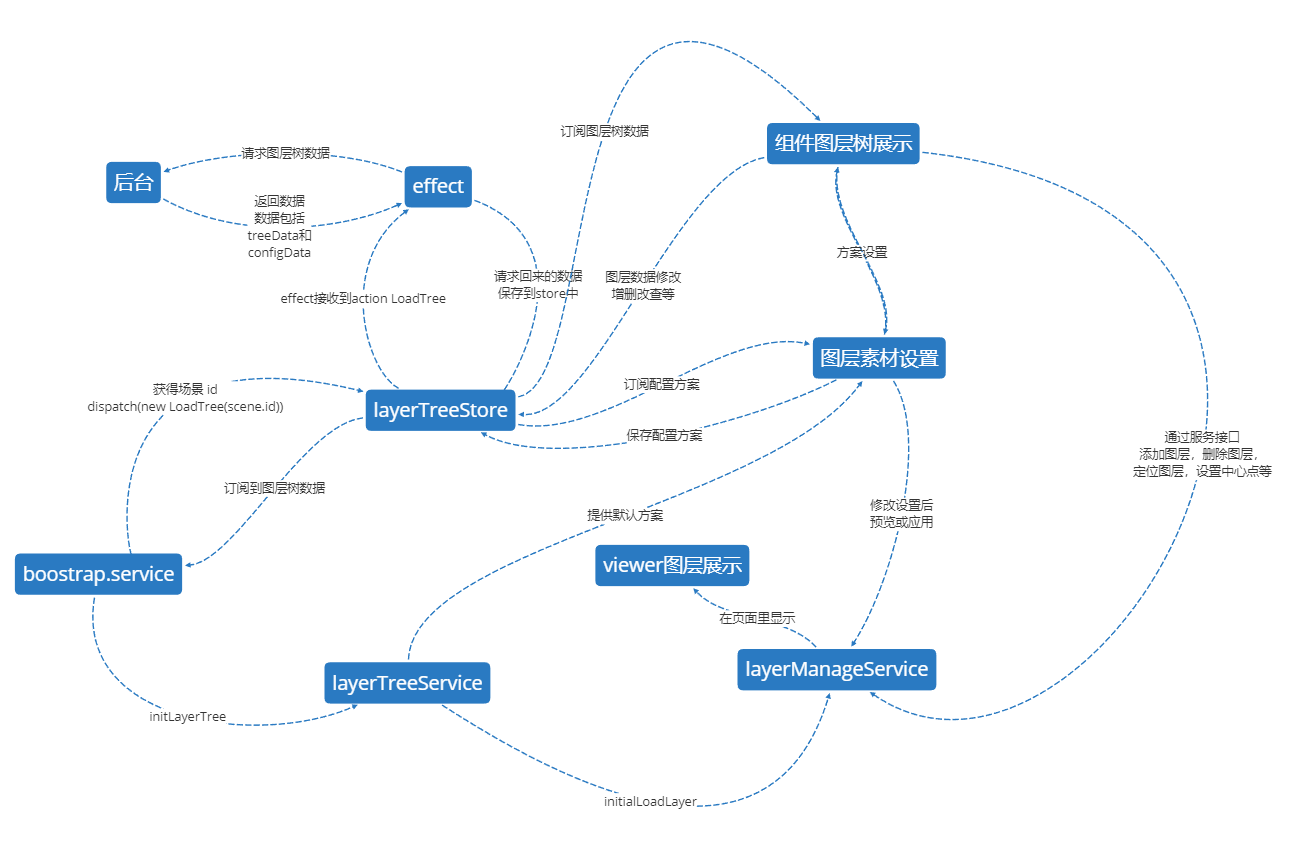
图层树重构设计：

**总体思路：**



将图层树数据单独提取出来，主要数据结构为

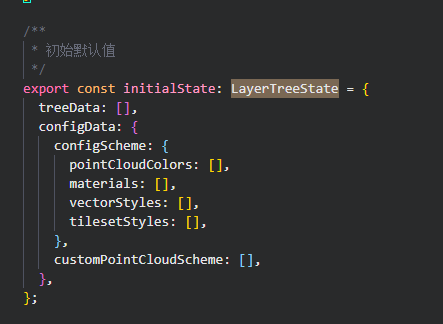


图1-1

通过数据与页面串联起来



图1-2

主要包含三大部分：

1. layerManageComponent 即图1-2的A部分  
   此部分主要功能为：  
   渲染图层树形结构，操作树形结构，包括增删改，过滤，拖拽，添加图层素材，以及素材的定位，设置，中心点移动等功能  
     
   （2）layerTreeStroe 即图1-1部分

此部分主要功能为：

获取服务器上场景已保存的图层树数据，并存储到store 里，供组件获取，并提供接口供组件对其数据进行修改，然后根据其他组件的订阅对其发送数据

（3）layerManageService 即图1-2的B部分

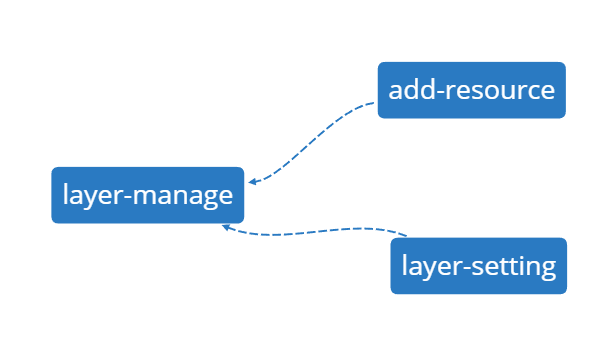
此部分主要功能为：

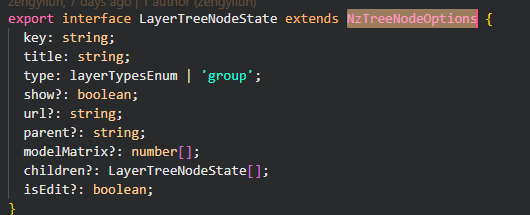
根据layerManageComponent 和layerTreeStroe 提供的数据，对场景里的viewer里的layer进行设置或修改，包括添加图层，控制图层显隐，飞行定位，对图层的style进行修改，以及对layer的其他属性进行修改等

（4）layerTreeService 主要负责提供图层树一些数据处理的服务

详细说明如下：

(一)、layerManageComponent部分





图层树节点数据结构

① 图层树过滤功能

由于项目中使用的UI库是ng-zorro 7.x版本，支持 Angular ^7.0.0 版本，而此版本中，组件nz-tree的过滤功能，存在严重的bug，并且无法升级，因为会对项目的其他依赖产生影响，所以图层树的过滤功能是通过自己过滤数据实现的。

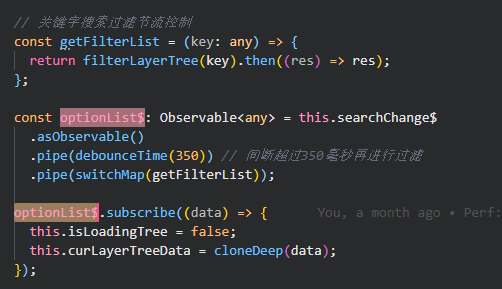
因此图层树的数据主要由2个字段控制：  
1. layerTreeData 保存整个图层树的完整数据

1. curLayerTreeData 保存过滤后的图层树过滤（当未搜索过滤时，与1是一样的）



输入框进行搜索，停止输入350毫秒后进行数据过滤

searchChange$ = new BehaviorSubject(null);



使用rxjs 的 BehaviorSubject 存储最后一条数据，之所以使用BehaviorSubject ，是因为BehaviorSubject是可以存储最后一条数据或者初始默认值的， 所以无论订阅者什么时候订阅到数据源optionList$上, 都能接收到数据。我们只需要拿到350毫秒后不再输入的最后一次关键字进行过滤即可

通过获取到的key， 如‘天马’

使用封装的方法filterTreeDataByTitle

基于layerTreeData为源数据，将其进行递归，递归思路如下：

条件： 节点的title包含‘天马’

定义新的节点集 [ ]

将layerTreeData进行遍历，如果节点符合条件 ，直接加入到新的节点集，若其存在 children，并接着处理其 children

如果节点自己不符合条件，需要根据子集来判断它是否将其加入新节点集，如果子孙集中有符合要求的节点，加入，否则，不加入（因为整个子集都没有符合条件的）

将返回的数据赋给curLayerTreeData，这样，就实现了图层树的过滤功能

当清楚过滤输入框或输入框为空时，将layerTreeData的值复制给curLayerTreeData

由于图层树的其他功能，如添加同级组节点，添加子组节点，添加子素材，删除节点，批量删除，修改名称，拖拽，复选框（即图层显隐）等功能都会基于过滤状态和非过滤状态2种情况进行处理，因此此处先进行说明

每次操作图层树都会通过dispatch修改store 里的layerTree数据

1. 未过滤状态

进行操作后，curLayerTreeData的值将会进行改变，此时curLayerTreeData就是所有的完整的树结构，因此直接this.store.dispatch(new SetLayerTreeData(this.curLayerTreeData))

1. 过滤状态

进行操作后，curLayerTreeData的值也会进行改变，但此时curLayerTreeData不是所有的完整的树结构，要根据节点的操作，去更新总的数据layerTreeData的树结构。比如过滤情况下删除一个节点，删除key为123的节点，此时不仅curLayerTreeData需要改变（curLayerTreeData是页面显示的实时数据），layerTreeData也要进行改变，此时需要遍历layerTreeData，将key为123的节点删除，删除后的layerTreeData才是完整的所有数据，然后再通过

this.store.dispatch(new SetLayerTreeData(layerTreeData))更新store里的layerTree数据

② 图层树节点选中功能

默认情况下，图层树只能选中一个节点，将该节点存储在currentSelectKey数组中，如

currentSelectKey = [ ‘ 123 ’ ]， 选中的作用为，当选中时，点击顶部的添加分组，将添加为选中的这个节点的下一个兄弟节点，当没选中一个节点时，将添加为最后一个节点

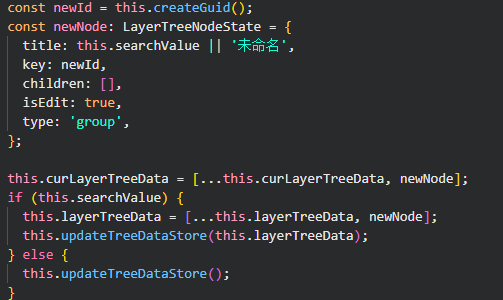
特殊情况（开启批量选中功能），可以选中多个节点，可以进行批量删除，此时无法添加分组

③ 图层树添加分组功能

同样都分了未过滤和过滤2种情况

1. 未选中节点，即currentSelectKey = [ ]

此时在树数据最后添加节点



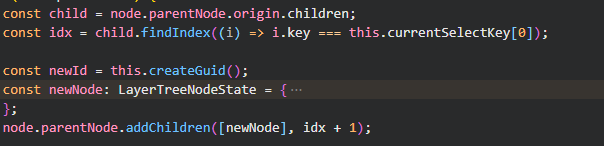
1. 有选中一个节点（选中多个节点时无法添加分组），如currentSelectKey = [ ‘ 123‘ ]



获取整个选中的node

ⅰ. 如果node.parentNode存在，即node不是首层数据，此时可以使用nzTreeComponent

的NzTreeNode的addChildren方法添加分组节点



找到key为123的节点的位置，在它后面插入新节点



当处于过滤状态时，需要更新layerTreeData数据，遍历整棵树，当key === 123 时，插入新数据，并更新到store

ⅱ. 如果node.parentNode不存在，即node是首层数据



此时找出key === 123 的节点位置，并在它后面插入新节点即可

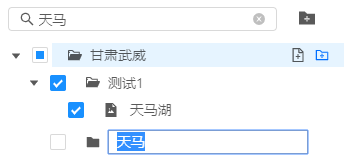
同样，当处于过滤状态时，需更新layerTreeData数据，更新方法如上，只需将curLayerTreeData改成layerTreeData

添加后，此时处于编辑状态，即isEdit为true，此时名称为输入框状态，需要获取该输入框并且聚焦和全选文字（focus和select）

名称默认为‘未命名’，当处于过滤状态时，名称默认为过滤的关键字

④ 图层树添加子分组功能

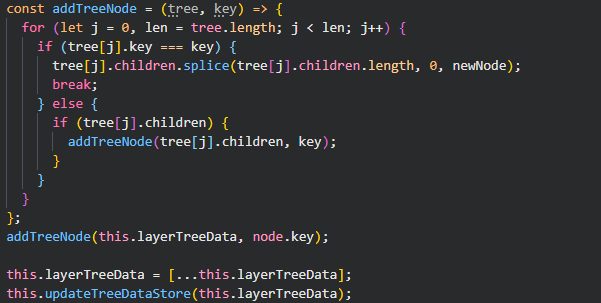
图层树添加分组后，该分组有添加子分组功能



添加子分组说明存在父节点，直接使用node.addChildren([newNode])添加

同样，当处于过滤状态时，需更新layerTreeData数据，遍历后插入

方法如下：



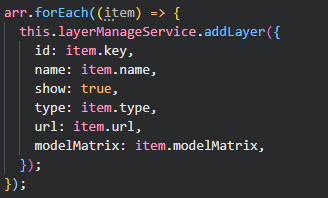
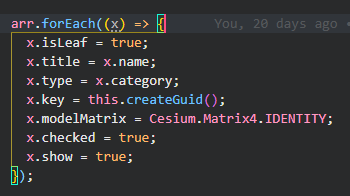
⑤ 图层树分组添加素材功能



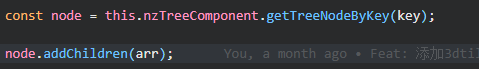
通过组件add-resource添加素材（add-resource组件后面再详细讲解）

保存curParentId ，打开添加素材弹窗

关闭添加素材弹窗后，获取已选择的素材数据数组

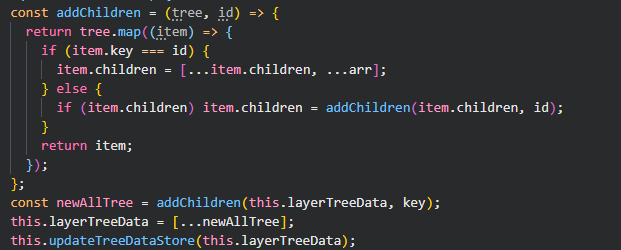


格式化素材数据，然后通过layerManageService添加图层（后面再讲）



获取节点，并添加子节点（即添加素材）

同样，当处于过滤状态时，需更新layerTreeData数据，遍历后插入



以上，图层的添加操作已经完成说明

⑥ 图层树节点修改名称功能



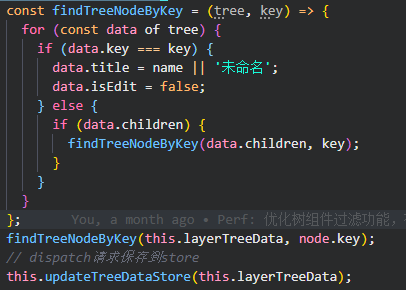
改变节点isEdit为true，输入框focus及select选中所有

此时还有一个功能点是，需要改变图层树的可拖拽状态（后面拖拽功能讲），当节点为可编辑状态时，改变图层树为不可拖拽

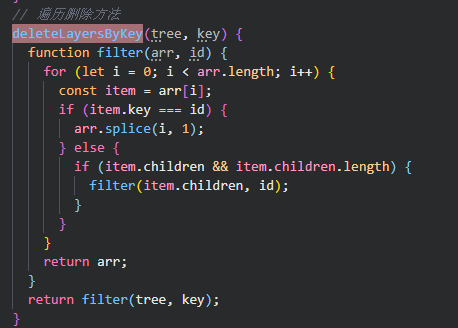
完成输入后，可通过回车键enter或者点击输入框外面区域（即失焦）触发修改

同样，当处于过滤状态时，需更新layerTreeData数据，遍历后修改名称

方法如下：



⑦ 图层树节点删除功能

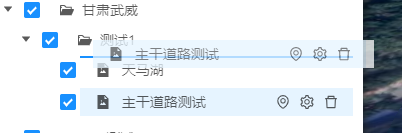


根据要删除的key，遍历返回删除后的数据，同样分过滤与未过滤将新数据保存到store中

除此之外，还要判断节点是否为素材，即node.isLeaf 是否为true，将素材从图层layer中删除，使用this.layerManageService.removeLayer（后面Service部分讲），同时删除该key在图层树store中的配置方案（后面store部分会将）

1. 如果该节点为叶子节点，即为图层素材，直接调用service进行layer删除
2. 如果该节点不为叶子节点，即为图层组，遍历其子孙节点下的所有叶子节点，即所有图层素材，获取所有图层素材，并依次调用service从layer中删除。

⑧ 图层树拖拽功能



图层树的拖拽功能，nz-tree组件自身存在bug，bug为拖拽后，有部分数据仍然保存，因此这部分也进行了改造，不用组件自带的方法，而是通过代码自己整理数据

拖拽节点可以获取2个 数据：

dragNode：拖动的节点

node： 拖拽到的关联的节点

ⅰ. 如果dragNode.parentNode存在

可以获取到dragNode.parentNode.origin，将其赋值给newNode

如果newNode.key 跟 node.key 相等时，图层树拖拽机制会刷新newNode节点，包括它的子孙节点都会更新，此时只需要遍历删除原始数据的dragNode节点，然后遍历将newNode的值赋值给相同key的节点即可。



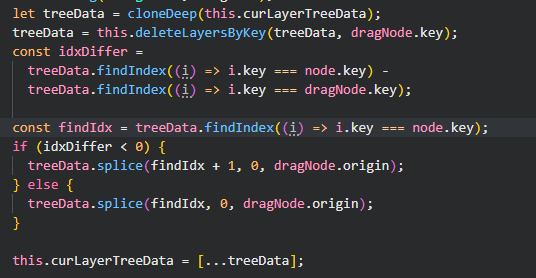
如果newNode.key 跟 node.key 不相等时，具体操作为：将叶子节点，拖到与它父级节点同一级的位置上，nz-tree的刷新机制有bug，newNode的children中，那个被拖动的节点的父节点，不会自动删除拖动的节点，此时需要手动过滤删除该节点，具体方式如下：



同样，当处于过滤状态时，需更新layerTreeData数据，遍历后更新数据

ⅰ. 如果dragNode.parentNode不存在

说明拖动后改节点处于树的最外层，处理方法如下：



先在原始树结构上删除被拖动的节点，然后判断节点是拖放到目标节点的前面还是后面，通过数组splice方法插入即可。

同样，当处于过滤状态时，需更新layerTreeData数据

let allTreeData = cloneDeep(this.layerTreeData);

allTreeData = this.deleteLayersByKey(allTreeData, dragNode.key);

克隆一份原始数据，删除被拖拽的节点

const addNode = this.findTreeNodeByKey(this.layerTreeData,dragNode.key)

从原始总数据获取拖拽的节点

然后跟上面一样，判断位置，通过splice方法插入。

⑨ 图层树添加素材功能

图层分组节点有添加素材的功能，添加素材使用组件add-resource，页面如下：

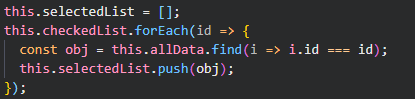


弹窗左边为搜索条件，右边为搜索结果

这里需要根据搜索关键字获取所有分类数据，然后可以根据分类类型进行搜索，并且需要显示各分类的总量，以及选中的数量

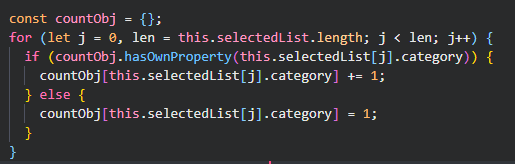
弹窗右边为搜索结果表格，可以点击某一行选择该素材，并实时更新下方的已选数据，可清空所有已选数据。当搜索关键字更改时，重置已选素材，重新选择。

选中或取消选中素材时，更新已选数据，获取所有选中的key

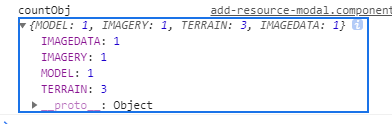


从总的数据里筛选出selectedList

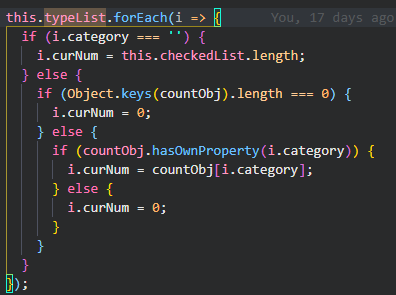
然后从selectedList中获取一个以素材类型为键，类型数量为值的obj



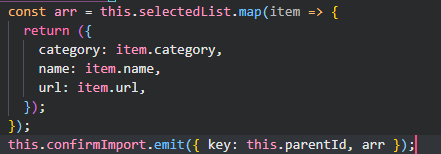
如：



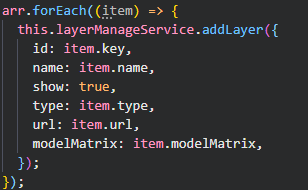
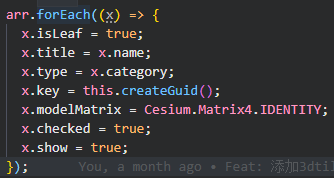
然后更新搜索条件的每种类型的选中数量即可



最后，选完素材后，点击导入，格式化素材，将selectedList数据传给layer-Manage



最后，在layer-manage组件中接收从add-resource组件传来的数据

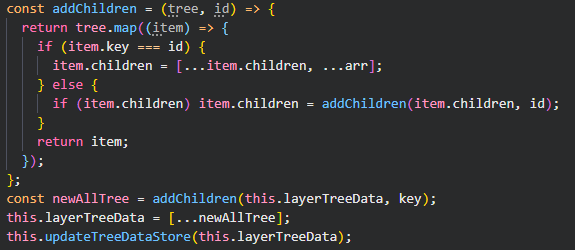


格式化素材数据，格式化成加载图层所需的格式，通过layerManageService.addLayer将素材依次添加到layer中（后面layerManageService部分再讲）



获取图层树的父节点，添加素材节点即可。

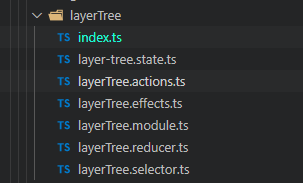
同样，当处于过滤状态时，需更新layerTreeData数据，遍历后合并数组即可



⑩ 图层树素材的定位，设置，中心点移动等功能

这些有关图层素材的功能都与layerManageService有关，具体放到后面一起讲解

(二)、layerTreeStroe部分



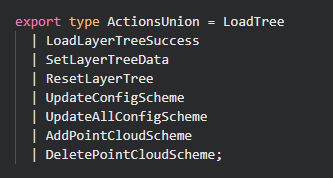
Store存储的数据结构如下



treeData存储图层树的数据，即layerManageComponent 的 layerTreeData

configData保存图层素材的配置方案（configScheme）和其他配置数据，目前有点云数据自定义方案（customPointCloudScheme）

action文件提供一些方法去更新store的LayerTreeState数据



effect文件的作用是，当打开场景时，获取到场景id后，会dispatch(new LoadTree(id))的action，effect监听到后，根据id请求后台服务，获取LayerTreeState的数据，并保存到store中，其他服务订阅到LayerTreeState变化后，作出其他操作，获取图层素材，根据配置方案格式化素材，然后通过layerManageService加载素材等。

effect文件的作用是，将数据分离，layerTree的数据只由它去获取。

(三)、layerManageService部分

layerManageService提供图层的各种方法，如

addLayer： 新增图层的方法

removeLayer： 移除图层的方法

flyToLayer： 飞行定位到图层

setLayerCenter: 设置中心点

setLayerShow: 设置图层显隐

.

.

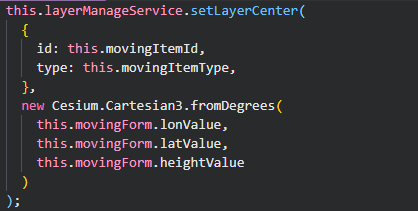
.

以及其他设置图层属性配置的方法（设置点云图层样式、设置 3D Tiles 的参数、设置模型材质参数、设置矢量图层样式等）

layerManageComponent 中，图层树素材的定位，设置，中心点移动等功能

1、定位：通过传递图层的id、type、show即可

2、中心点移动：通过以下方法



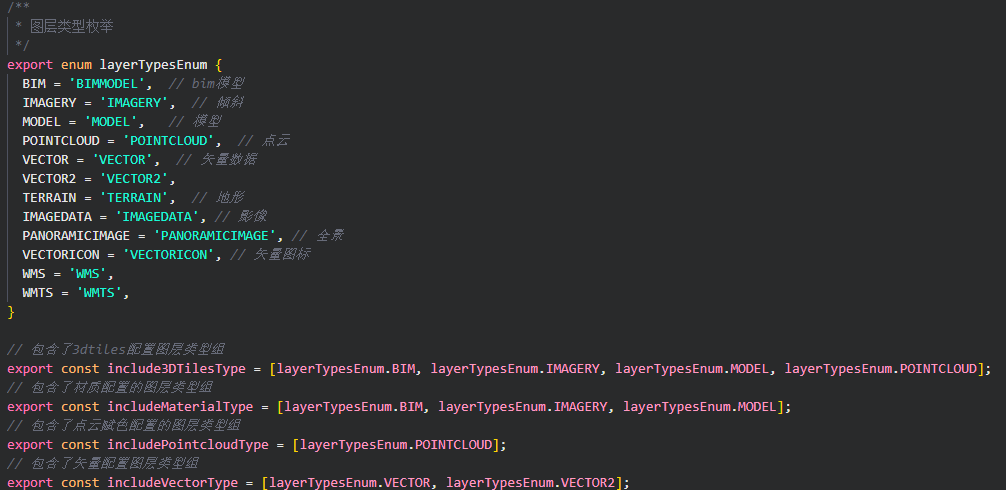
还需将移动后的中心点信息保存到图层树数据中，

通过layerManageService.getModelMatrix获取modelMatrix，然后遍历保存到该node中

1. 设置：

根据需求文档



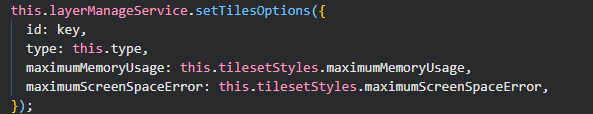


枚举图层类型，对各设置类型进行分类

根据不同类型组拥有的类型弹出对应的设置弹窗

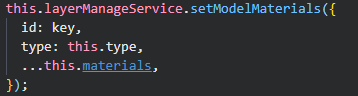
（1）、3D Tiles选项设置





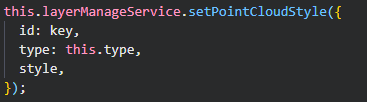
（2）、模型材质设置



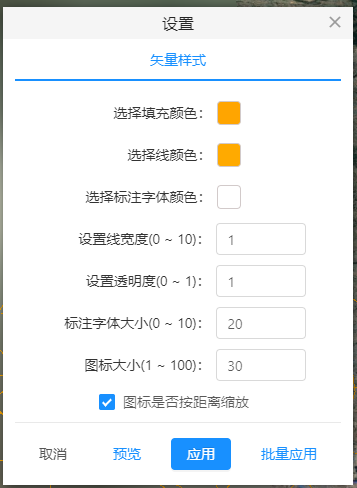


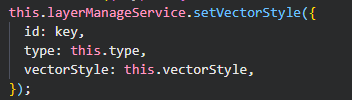
（3）、点云颜色设置





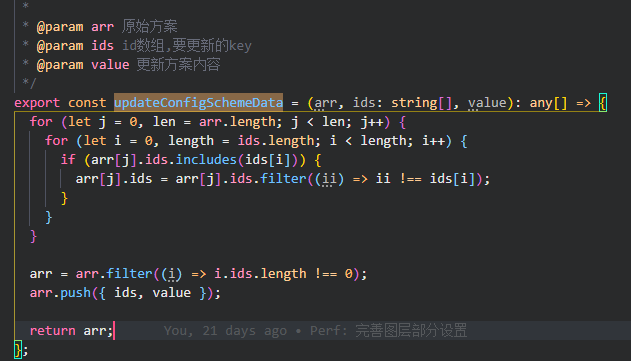
（4）、矢量样式设置





**设置的关键在于** 要从store里获取配置方案，若存在方案，则将方案的值赋值，若不存在，则要获取layerTreeService中获取该类型图层的默认方案

当修改方案后，点击应用或者批量应用，通过封装方法，修改原先store里的方案数据



存方案的方式如下：

1.该id不存在任何一个方案中，则新增一条数据，格式为{ ids: ['123'], value: {xxx}}

2. 该id存在一个方案中，且该方案只有该id使用，则直接替换改方案的值

3. 该id存在一个方案中，但该方案不止该id使用，则先将该方案中去除该id，然后新增一条新的方案

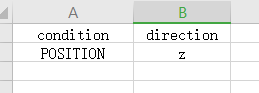
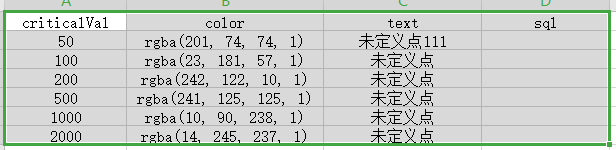
点击取消，或者关掉弹窗后，将获取该图层原先的方案值，然后进行设置

1. 、点云颜色设置的方案导出功能，以及自定义方案导入功能

导出方案： 将方案中的信息导出

包括临界值（criticalVal），颜色（color），文字说明（text），sql（为以后拓展留的字段），

条件（condition），方向（direction）



导入自定义方案：

只支持Excel导入方案，将导入的方案存到store的customPointCloudScheme中

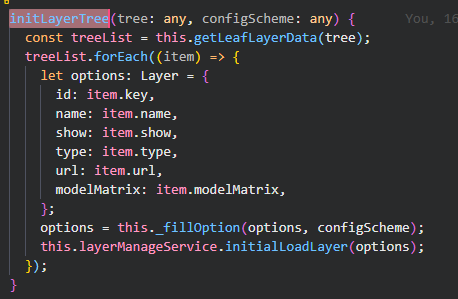


自定义方案可以删除

赋色方案本身的数据无法修改，选择某赋色方案，是继承该方案的数据

(四)、layerTreeService部分

该部分主要负责图层树的一些数据处理，比如，初始化场景时，获取图层树数据并根据store的方案配置格式化图层数据，然后通过layerManageService添加图层



其中getLeafLayerData是获取所有的叶子节点，即所有的图层素材

\_fillOption是根据配置方案填充素材属性的方法，通过这2步后，素材数据是一个完整的layer数据，直接通过layerManageService.initialLoadLayer添加图层

还有getOriginData方法，即当素材没有方案配置时，获取平台默认的配置数据

默认数据如下：

