#### 设计器包含主要功能

1）资源维护

1. 组件选择器
2. 组件树查看功能
3. 画布
4. 工具栏

保存

撤销

恢复

删除

对齐功能

生成代码

1. 属性编辑栏

数据集定义

事件定义

包含定义与后台交互的规则、访问数据集的规则，访问事件的规则

其他基本属性定义

1. 快捷键定义栏
2. 使用说明书

#### 设计器原理：



设计器主要关注点

数据集定义：

动作定义：

组件定义：

样式定义(目前没有考虑)：

四者之间的关系：



组件只是数据状态的展示，所有的展示通过操作数据集的变化来反映。

##### 组件树数据结构 （Map 对象）

{

Key: 组件ID

Value:compent

}

##### 组件数据结构（ object 对象）

{

Id:’’组件ID

,type:’’

, props:{

}

,event:{

}

,binding:’’--绑定的数据集

}

##### 数据集的数据结构（Map 对象）

{

Key : ’’-数据集名称

Value : {

Name:’’ --数据集名称

Describe:’’数据集描述信息

Type:’’ -- 数据集类型，arry,object,Boolean,string,number

initData: object--初始值,

data: --操作的值

}

}

##### 动作集的数据结构（Map 对象）

{

Key:动作名称

Value{

Name:’’动作名称

Describe:’’ 动作描述

parameters:’’传入参数，多个参数以逗号分隔

body:’’函数体

action:new Function(this.parameters,this.body)生成的js函数

}

}

##### 保存的数据结构

保存的数据结构为xml结构。举例

<?xml version="1.0" encoding="utf-8"?>

<root>

<Slide> --元件组定义

<parent>null</parent>

<type>Slide</type>

<id>231b8513-8c84-454f-ba1e-4161e132aa8b</id>

<props>

<style><![CDATA[{"height":"100%","width":"100%"}]]></style>

<transition><![CDATA[["slide"]]]></transition>

</props>

<event><![CDATA[{"init":"init"}]]></event>

<binding>undefined</binding>

<children>

<Button>

<parent>231b8513-8c84-454f-ba1e-4161e132aa8b</parent>

<type>Button</type>

<id>13cbfb6f-4e58-4ce0-ae34-a08b23bd7fba</id>

<props>

<style><![CDATA[{"width":100,"height":50,"left":121,"top":133}]]></style>

<label><![CDATA["按钮"]]></label>

<size><![CDATA["large"]]></size>

<type><![CDATA["primary"]]></type>

<plain><![CDATA[false]]></plain>

<loading><![CDATA[false]]></loading>

<disabled><![CDATA[false]]></disabled>

<icon><![CDATA[""]]></icon>

<nativeType><![CDATA["button"]]></nativeType>

</props>

<event><![CDATA[{"click":""}]]></event>

<binding>undefined</binding>

<children/>

</Button>

</children>

</Slide>

<Actions>--动作定义

<child>

<name>init</name>

<describe/>

<body><![CDATA[]]></body>

<paramters><![CDATA[]]></paramters>

<action><![CDATA[function action() {}]]></action>

</child>

</Actions>

<DataSet>

</DataSet>

</root>

##### 开发使用依赖的模块说明

Mobx ：状态管理的库

React-dnd ：拖拽组件

Xmljs: xml解析

###### 拖拽实现原理：

基本用法

1. 把应用的根组件包装在 DragDropContext 中
2. 把可以拖拽的组件包装在 DragSource 中
   1. 设置 type，DragSource 和 DropTarget 必须指定 type。只有在 type 相同的情况下，DragSource 才能放到 DropTarget 中
   2. 设置 spec，让组件可以响应拖拽事件
   3. 设置 collect，把拖拽过程中需要信息注入组件的 props
3. 把可以接受拖拽的组件包装在 DropTarget 中
   1. 设置 type，DragSource 和 DropTarget 必须指定 type。只有在 type 相同的情况下，DragSource 才能放到 DropTarget 中
   2. 设置 spec，让组件可以响应拖拽事件
   3. 设置 collect，把拖拽过程中需要信息注入组件的 props
4. 完

实例：

1. 将应用根包装在DragDropContxt中

import { DragDropContextProvider } from 'react-dnd'

import HTML5Backend from 'react-dnd-html5-backend'

class Desginer extends Component {

render(){

return (

<DragDropContextProvider backend={HTML5Backend}> // HTML5Backend不在在触摸屏下使用

<Canvas></Canvas>

</DragDropContextProvider>

</div>

)}

);

}

2、可拖拽元件包装在Dragsource中

import { DragSource } from 'react-dnd'

//定义spec

const boxSource = {

  beginDrag(props) {

    return {

      elementType: props.elementType,

    }

  },

  endDrag(props, monitor) {

const item = monitor.getItem()

const dropResult = monitor.getDropResult()//获取drop目标

     if (dropResult) {

//获取drop目标后执行想做的操作

}

)

   }

  },

}

@DragSource("element-types",

boxSource, (connect, monitor) => ({

  connectDragSource: connect.dragSource(),

  isDragging: monitor.isDragging(),

}))

class ElementItem extends Component {

/\*\*增加属性\*\*/

static propTypes = {

connectDragSource: PropTypes.func.isRequired,

    isDragging: PropTypes.bool.isRequired,

elementType: PropTypes.string.isRequired,

scale: PropTypes.number

};

 const { connectDragSource } = this.props

return connectDragSource(

<li>

文本框

</li>

);

}

3、把可以接受拖拽的组件包装在 DropTarget 中

import { DropTarget } from 'react-dnd'

//设置spec

const boxTarget = {

  drop(props,monitor) {

if(monitor.getDropResult()){

return {

id: monitor.getDropResult().id,

};

}

return { id: props.index}

},

}

@DropTarget("element-types", boxTarget, (connect, monitor) => ({

  connectDropTarget: connect.dropTarget(),

  isOver: monitor.isOver(),

  canDrop: monitor.canDrop(),

}))

class Slide extends Component {

//定义装饰器

static propTypes = {

scale: PropTypes.number.isRequired,

connectDropTarget: PropTypes.func.isRequired,

    isOver: PropTypes.bool.isRequired,

    canDrop: PropTypes.bool.isRequired,

};

render() {

const { canDrop, isOver, connectDropTarget } = this.props;

return connectDropTarget(

<div canDrop={canDrop} isOver={isOver} >

</div>

);

}

}

###### 状态管理实现原理：

基本用法

1、定义页面数据和相关的数据操作的一个store。

2、定义一个Provider 组件可以将store通过react的context机制传递给子组件

3、将根组件包裹在Provider中

3、使用static propTypes 使子组件获取 store。

实例

1、定义store

export default class Store{

@observable number = 1;//

Onclick()

{

this.number++;

}

}

1. 定义Provider

后期需要的内容

1）权限

2）路由

3）静态文件的存储