ggeditor

ggeditor

内置事件

内置事件可以分为这么几类：

1. 官方自定义动作事件
2. 编辑器事件
3. 通用事件
4. 节点事件
5. 边事件
6. 画布事件

以上为可点击链接，链接至 ggeditor 的 github 仓库。

均有备注，解释相当详细。

官方自定义动作事件

```
export enum GraphCustomEvent {
  /** 调用 add / addItem 方法之前触发 */
  onBeforeAddItem = 'beforeadditem',
  /** 调用 add / addItem 方法之后触发 */
  onAfterAddItem = 'afteradditem',
  /** 调用 remove / removeItem 方法之前触发 */
  onBeforeRemoveItem = 'beforeremoveitem',
  /** 调用 remove / removeItem 方法之后触发 */
  onAfterRemoveItem = 'afterremoveitem',
  /** 调用 update / updateItem 方法之前触发 */
  onBeforeUpdateItem = 'beforeupdateitem',
  /** 调用 update / updateItem 方法之后触发 */
  onAfterUpdateItem = 'afterupdateitem',
  /** 调用 showItem / hideItem 方法之前触发 */
  onBeforeItemVisibilityChange = 'beforeitemvisibilitychange',
  /** 调用 showItem / hideItem 方法之后触发 */
  onAfterItemVisibilityChange = 'afteritemvisibilitychange',
  /** 调用 setItemState 方法之前触发 */
  onBeforeItemStateChange = 'beforeitemstatechange',
```

```
  /** 调用 setItemState 方法之后触发 */
  onAfterItemStateChange = 'afteritemstatechange',
  /** 调用 refreshItem 方法之前触发 */
  onBeforeRefreshItem = 'beforerefreshitem',
  /** 调用 refreshItem 方法之后触发 */
  onAfterRefreshItem = 'afterrefreshitem',
  /** 调用 clearItemStates 方法之前触发 */
  onBeforeItemStatesClear = 'beforeitemstatesclear',
  /** 调用 clearItemStates 方法之后触发 */
  onAfterItemStatesClear = 'afteritemstatesclear',
  /** 布局前触发。调用 render 时会进行布局，因此 render 时会触发。或用户主动调用图的 layout 时触发 */
  onBeforeLayout = 'beforelayout',
  /** 布局完成后触发。调用 render 时会进行布局，因此 render 时布局完成后会触发。或用户主动调用图的 layout 时布局完成后触发 */
  onAfterLayout = 'afterlayout',
  /** 连线完成之前触发 */
  onBeforeConnect = 'beforeconnect',
  /** 连线完成之后触发 */
  onAfterConnect = 'afterconnect',
}
```

编辑器事件

```
export enum EditorEvent {
  /** 调用命令之前触发 */
  onBeforeExecuteCommand = 'onBeforeExecuteCommand',
  /** 调用命令之后触发 */
  onAfterExecuteCommand = 'onAfterExecuteCommand',
  /** 改变画面状态触发 */
  onGraphStateChange = 'onGraphStateChange',
  /** 改变标签状态触发 */
  onLabelStateChange = 'onLabelStateChange',
}
```

通用事件

```ts
export enum GraphCommonEvent {
  /** 单击鼠标左键或者按下回车键时触发 */
  onClick = 'click',
  /** 双击鼠标左键时触发 */
  onDoubleClick = 'dblclick',
  /** 鼠标移入元素范围内触发，该事件不冒泡，即鼠标移到其后代元素上时不会触发 */
  onMouseEnter = 'mouseenter',
  /** 鼠标在元素内部移到时不断触发，不能通过键盘触发 */
  onMouseMove = 'mousemove',
  /** 鼠标移出目标元素后触发 */
  onMouseOut = 'mouseout',
  /** 鼠标移入目标元素上方，鼠标移到其后代元素上时会触发 */
  onMouseOver = 'mouseover',
  /** 鼠标移出元素范围时触发，该事件不冒泡，即鼠标移到其后代元素时不会触发 */
  onMouseLeave = 'mouseleave',
  /** 鼠标按钮被按下（左键或者右键）时触发，不能通过键盘触发 */
  onMouseDown = 'mousedown',
  /** 鼠标按钮被释放弹起时触发，不能通过键盘触发 */
  onMouseUp = 'mouseup',
  /** 用户右击鼠标时触发并打开上下文菜单 */
  onContextMenu = 'contextmenu',
  /** 当拖拽元素开始被拖拽的时候触发的事件，此事件作用在被拖曳元素上 */
  onDragStart = 'dragstart',
  /** 当拖拽元素在拖动过程中时触发的事件，此事件作用于被拖拽元素上 */
  onDrag = 'drag',
  /** 当拖拽完成后触发的事件，此事件作用在被拖曳元素上 */
  onDragEnd = 'dragend',
  /** 当拖曳元素进入目标元素的时候触发的事件，此事件作用在目标元素上 */
  onDragEnter = 'dragenter',
  /** 当拖曳元素离开目标元素的时候触发的事件，此事件作用在目标元素上 */
  onDragLeave = 'dragleave',
  /** 被拖拽的元素在目标元素上同时鼠标放开触发的事件，此事件作用在目标元素上 */
  onDrop = 'drop',
  /** 按下键盘键触发该事件 */
  onKeyDown = 'keydown',
  /** 释放键盘键触发该事件 */
```

```
  onKeyUp = 'keyup',
  /** 当手指触摸屏幕时候触发，即使已经有一个手指放在屏幕上也会触发 */
  onTouchStart = 'touchstart',
  /** 当手指在屏幕上滑动的时候连续地触发。在这个事件发生期间，调用 preventDefaul
t() 事件可以阻止滚动。 */
  onTouchMove = 'touchmove',
  /** 当手指从屏幕上离开的时候触发 */
  onTouchEnd = 'touchend',
}
```


节点事件

```
export enum GraphNodeEvent {
  /** 鼠标左键单击节点时触发 */
  onNodeClick = 'node:click',
  /** 鼠标双击左键节点时触发 */
  onNodeDoubleClick = 'node:dblclick',
  /** 鼠标移入节点时触发 */
  onNodeMouseEnter = 'node:mouseenter',
  /** 鼠标在节点内部移到时不断触发，不能通过键盘触发 */
  onNodeMouseMove = 'node:mousemove',
  /** 鼠标移出节点后触发 */
  onNodeMouseOut = 'node:mouseout',
  /** 鼠标移入节点上方时触发 */
  onNodeMouseOver = 'node:mouseover',
  /** 鼠标移出节点时触发 */
  onNodeMouseLeave = 'node:mouseleave',
  /** 鼠标按钮在节点上按下（左键或者右键）时触发，不能通过键盘触发 */
  onNodeMouseDown = 'node:mousedown',
  /** 节点上按下的鼠标按钮被释放弹起时触发，不能通过键盘触发 */
  onNodeMouseUp = 'node:mouseup',
  /** 用户在节点上右击鼠标时触发并打开右键菜单 */
  onNodeContextMenu = 'node:contextmenu',
  /** 当节点开始被拖拽的时候触发的事件，此事件作用在被拖曳节点上 */
  onNodeDragStart = 'node:dragstart',
  /** 当节点在拖动过程中时触发的事件，此事件作用于被拖拽节点上 */
  onNodeDrag = 'node:drag',
```

```
  /** 当拖拽完成后触发的事件，此事件作用在被拖曳节点上 */
  onNodeDragEnd = 'node:dragend',
  /** 当拖曳节点进入目标元素的时候触发的事件，此事件作用在目标元素上 */
  onNodeDragEnter = 'node:dragenter',
  /** 当拖曳节点离开目标元素的时候触发的事件，此事件作用在目标元素上 */
  onNodeDragLeave = 'node:dragleave',
  /** 被拖拽的节点在目标元素上同时鼠标放开触发的事件，此事件作用在目标元素上 */
  onNodeDrop = 'node:drop',
}
```

边事件

```
export enum GraphEdgeEvent {
  /** 鼠标左键单击边时触发 */
  onEdgeClick = 'edge:click',
  /** 鼠标双击左键边时触发 */
  onEdgeDoubleClick = 'edge:dblclick',
  /** 鼠标移入边时触发 */
  onEdgeMouseEnter = 'edge:mouseenter',
  /** 鼠标在边上移到时不断触发，不能通过键盘触发 */
  onEdgeMouseMove = 'edge:mousemove',
  /** 鼠标移出边后触发 */
  onEdgeMouseOut = 'edge:mouseout',
  /** 鼠标移入边上方时触发 */
  onEdgeMouseOver = 'edge:mouseover',
  /** 鼠标移出边时触发 */
  onEdgeMouseLeave = 'edge:mouseleave',
  /** 鼠标按钮在边上按下（左键或者右键）时触发，不能通过键盘触发 */
  onEdgeMouseDown = 'edge:mousedown',
  /** 边上按下的鼠标按钮被释放弹起时触发，不能通过键盘触发 */
  onEdgeMouseUp = 'edge:mouseup',
  /** 用户在边上右击鼠标时触发并打开右键菜单 */
  onEdgeContextMenu = 'edge:contextmenu',
}
```

画布事件

```ts
export enum GraphCanvasEvent {
  /** 鼠标左键单击画布时触发 */
  onCanvasClick = 'canvas:click',
  /** 鼠标双击左键画布时触发 */
  onCanvasDoubleClick = 'canvas:dblclick',
  /** 鼠标移入画布时触发 */
  onCanvasMouseEnter = 'canvas:mouseenter',
  /** 鼠标在画布内部移到时不断触发，不能通过键盘触发 */
  onCanvasMouseMove = 'canvas:mousemove',
  /** 鼠标移出画布后触发 */
  onCanvasMouseOut = 'canvas:mouseout',
  /** 鼠标移入画布上方时触发 */
  onCanvasMouseOver = 'canvas:mouseover',
  /** 鼠标移出画布时触发 */
  onCanvasMouseLeave = 'canvas:mouseleave',
  /** 鼠标按钮在画布上按下（左键或者右键）时触发，不能通过键盘触发 */
  onCanvasMouseDown = 'canvas:mousedown',
  /** 画布上按下的鼠标按钮被释放弹起时触发，不能通过键盘触发 */
  onCanvasMouseUp = 'canvas:mouseup',
  /** 用户在画布上右击鼠标时触发并打开右键菜单 */
  onCanvasContextMenu = 'canvas:contextmenu',
  /** 当画布开始被拖拽的时候触发的事件，此事件作用在被拖曳画布上 */
  onCanvasDragStart = 'canvas:dragstart',
  /** 当画布在拖动过程中时触发的事件，此事件作用于被拖拽画布上 */
  onCanvasDrag = 'canvas:drag',
  /** 当拖拽完成后触发的事件，此事件作用在被拖曳画布上 */
  onCanvasDragEnd = 'canvas:dragend',
  /** 当拖曳画布进入目标元素的时候触发的事件，此事件作用在目标元素上 */
  onCanvasDragEnter = 'canvas:dragenter',
  /** 当拖曳画布离开目标元素的时候触发的事件，此事件作用在目标元素上 */
  onCanvasDragLeave = 'canvas:dragleave',
}
```

上下文

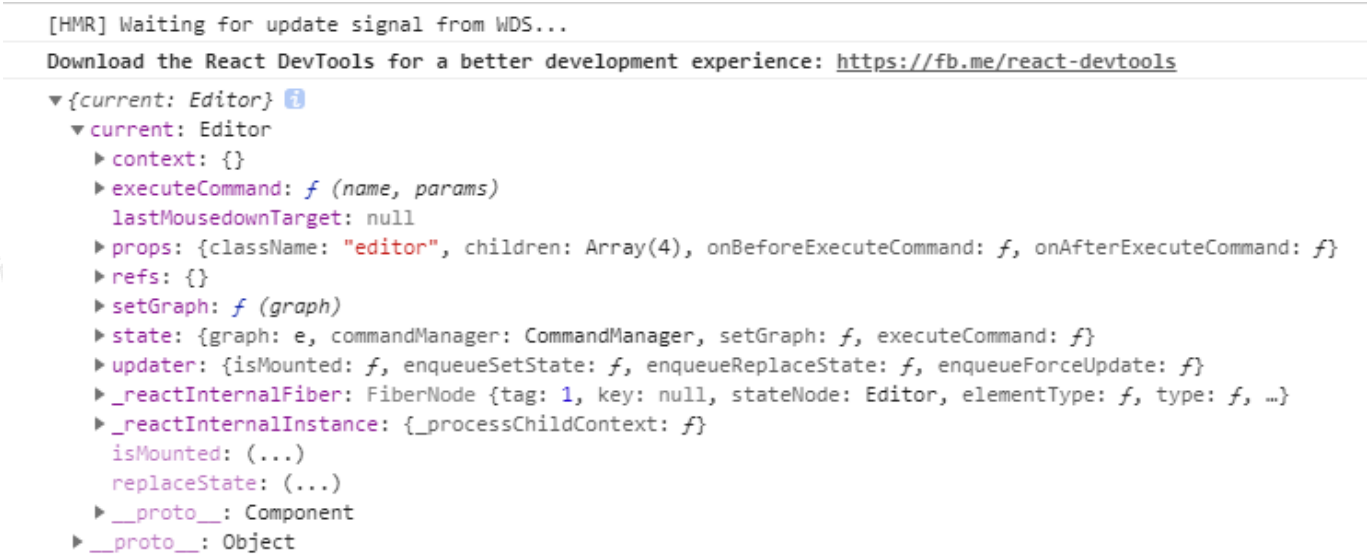
ggeditor 上下文具体概念，我也很难说清，我简单粗暴点理解的话，就是包含了 ggeditor 所有属性和方法的对象。

通过 ref 获取

我是这么获取的

```
class App extends Component {
  componentDidMount() {
    console.log(this.editorRef);
  }

  editorRef = React.createRef();

  render() {
    <GGEditor ref={this.editorRef}>
      ...
    </GGEditor>
  }

}

```

打印一下获取结果：



官方文档方式

> 还有另外一种方式，即官方文档上的方式。

ggeditor 官方文档：https://ggeditor.com/zh-CN/examples/editor/context

GGEditor（编辑器主控）

编辑器主控，是我自己取的名字，因为在 ggeditor 的所有组件，都必须要在该组件下。

```
import GGEditor from 'gg-editor';
```

```
<div className="app">
  <GGEditor className="editor">
        <Flow className="editorBd" />
        <EditableLabel />
    ...等组件
    </GGEditor>
</div>
```

GGEditor 就是我所说的编辑器主控。
需要注意的是：
GGEditor 的父元素（即：上一段示例 HTML 代码中，className 等于 app 的元素）的宽高需要确定。

以下是官方文档中的 CSS 代码，不保证有效，我更多的是使用 calc 函数计算宽高。

```
.app {
  display: flex;
  width: 100%;
  height: 100%;
  overflow: hidden;
}

.editor {
  display: flex;
  flex: 1;
  background-color: #f4f6f8;
}

```

```
14 .editorBd {
15   flex: 1;
16 }
```

Command（命令）

不在此做定义解释，举个例子来体会：Word 编辑器的撤销重做保存等操作，就是一个个命令。

触发模式

既然是命令，就需要人为触发，主要有两种触发模式：
一是手动触发，比如说点击了撤销、重做、保存按钮；
二是自动触发，比如说关闭 Word 文档时，Office 一般会做自动保存操作。
导入：

```
import { Command } from "gg-editor";
```

UI 层面的使用（手动触发）：

```
<Command name={EditorCommand.Undo}>
  <i className="fa fa-undo" />
</Command>
```

逻辑层面的使用（自动触发）：

```
待补充
```

props

> 组件可接收选项

name

这个 name 值是最重要，对应的必须是 ggeditor 中有效的内置命令或自定义命令，接收类型为字符串。

className

类名

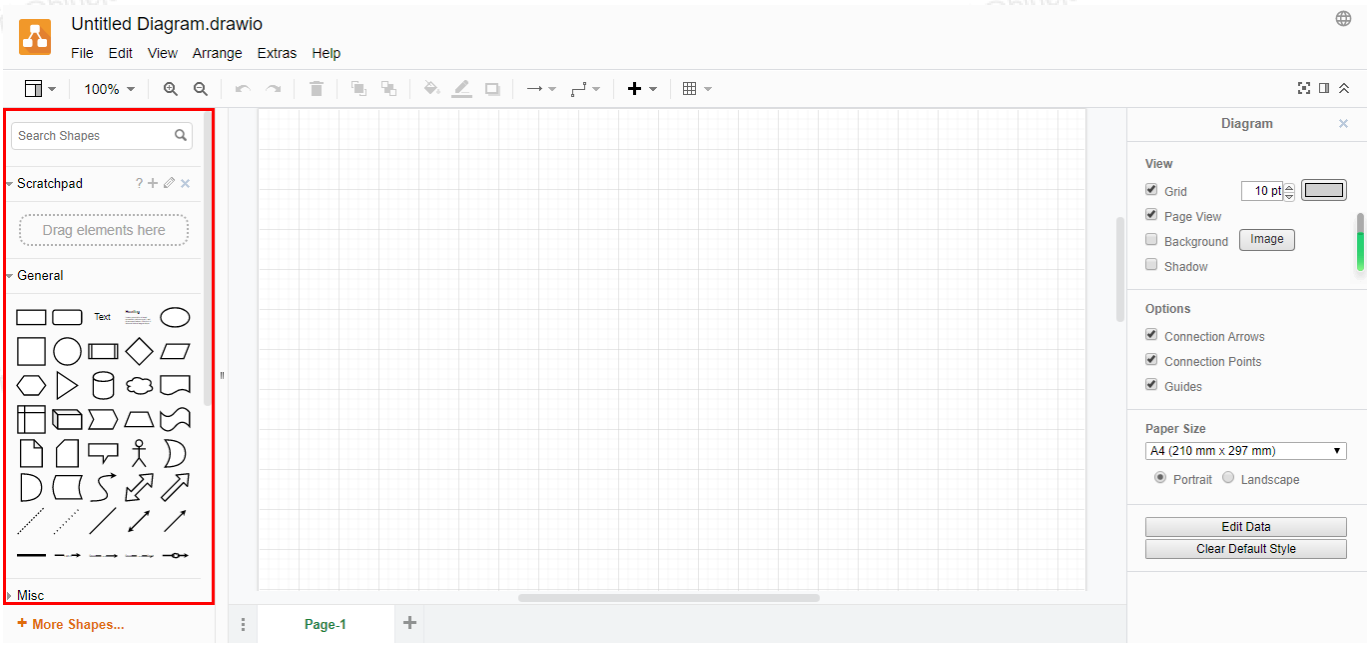
disabledClassName

禁用状态下的类名

Toolbar（工具栏）

ItemPanel（元素面板）

以下这，就是 ItemPanel（元素面板）。



导入：

```
import { ItemPanel, Item } from "gg-editor"
```

元素面板中，可以放 HTML，但需要被 Item 组件包裹起来。

```
<ItemPanel className="itempanel">
  <Item
    model={{
      type: "circle",
      size: 80,
      label: "circle",
    }}
  >
    <img
      src="https://gw.alicdn.com/tfs/TB1IRuSnRr0gK0jSZFnXXbRRXXa-110
```

```
  -112.png"
      width="90"
      height="90"
      draggable={false}
    />
  </Item>
</ItemPanel>
```

ItemPanel props

> 元素面板的可接收选项

除所有组件都有的 style 和 className 外，无另外的 props。

Item props

> 元素的可接收选项

type

描述该元素的类型，是 node（节点）还是 edge（边），默认为 node（节点）。

model

描述该节点元素的数据模型，数据模型的具体配置项，需要见 G6 文档，我筛选出了一些：

1. 内置节点类型说明
2. 元素配置项

需要注意的是：

1. Item 下的 img 需要将 draggable 设置为 false，禁止其可拖放。
2. Item 下的 img 需要设置一下宽高。
3. Item 的 type 可以不填，但 model 必填，因为它需要被注入到 Page（画布）的数据模型中。

Flow（流程图）

className，类名，同样需要设定好宽高，如果不知道怎么设置，可以像官方示例中这么写：

```
.editorBd {
  flex: 1;
}
```

data，数据

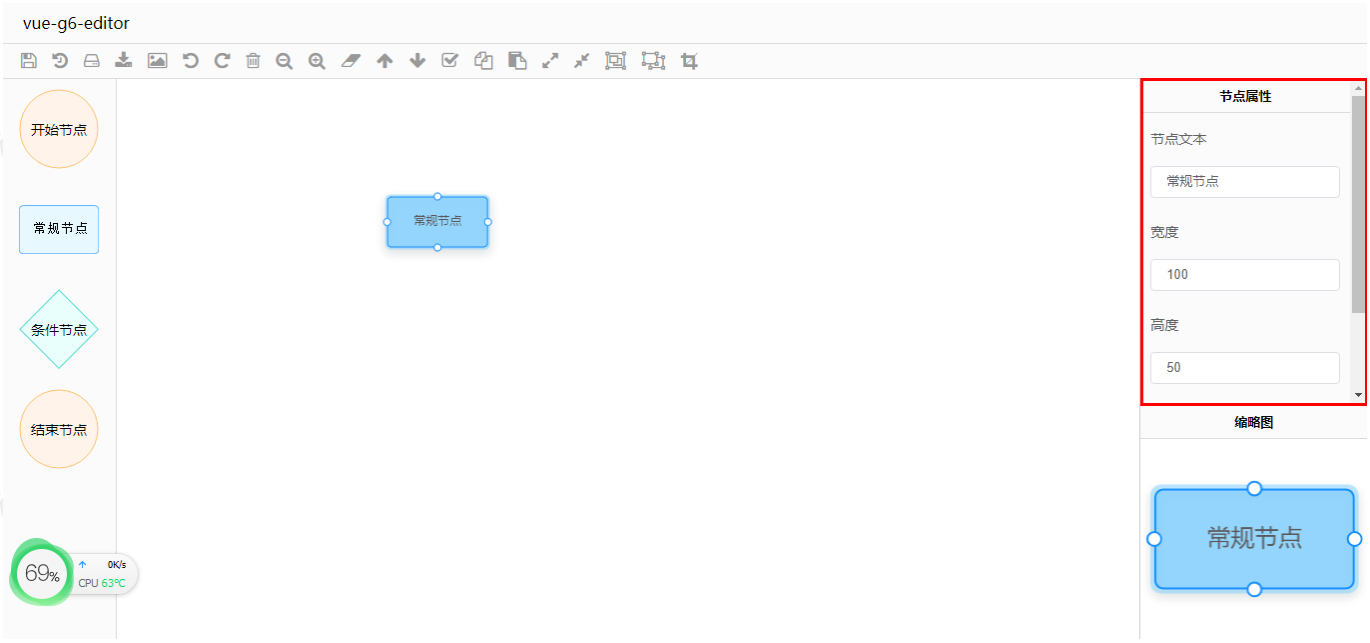
graphConfig，流程图配置，这个可以参考 G6 文档（https://g6.antv.vision/zh/docs/api/Graph#g6graph）

> G6 文档，个人感觉挺细的，但又感觉低内聚高耦合，各有各的好处吧，但确实需要一点时间来摸索熟悉各部分文档的关联关系，才能快速跳转到自己需要的内容部分。

补充一点：
ggeditor 基于 G6，除了支持 G6 的内置节点外，还提供了几个独有的内置节点，分别是 bizFlowNode、。

DetailPanel（详情面板）

以下就是，DetailPanel（详情面板）。



用过 g6-editor 同学可能知道，它不需要我们自己做在不同选中状态下，DetailPanel 的显示隐藏控制。而 ggeditor 提供的 DetailPanel 需要，由于 react 不同于 vue 的开发方式并提供了 v-if 这么一个方便的指令，所以我们可能需要基于 DetailPanel 二次封装，才能真正投入到业务开发中去。

ContextMenu（右键菜单）

与 g6-editor 相比的话，在 ContextMenu 也同样少了许多功能，需要自己做更多在在不同选中状态下的差异显示隐藏控制。

如果业务需要的右键菜单功能比较复杂的话，可能需要像 DetailPanel（详情面板）一样封装一层。

导入：

```
import { ContextMenu } from "gg-editor";
```

使用：

> 界面 UI 上搭配了 AntDesign 组件库使用

```
<ContextMenu
  type="node"
  renderContent={(item, position, hide) => {
    const { x: left, y: top } = position;
    return (
      <div style={{ position: 'absolute', top, left }}>
        <Menu mode="vertical" selectable={false} onClick={hide}>
          <Menu.Item>
            <Command name={EditorCommand.Copy}>
              复制
            </Command>
          </Menu.Item>
          <Menu.Item>
            <Command name={EditorCommand.Remove}>
              删除
            </Command>
          </Menu.Item>
        </Menu>
      </div>
    );
  }}
```

```
22 />
```

props
type（菜单类型）
有三个类型可选，分别是 canvas（画布）、node（节点）、edge（边）。
如果你选了 node（节点），那么右键菜单只在右键点击的是节点的情况下，才会显示，以此类推。

renderContent（菜单内容）
可接收选项的类型是函数。
函数需要返回 renderContent（菜单内容）的DOM。
函数有三个参数，你可以使用，分别是：

1. item（所选节点或边）
2. position（鼠标位置）
3. hide（右键菜单隐藏函数）

内置命令

相比 g6-editor，少了很多内置命令。

中文名称	英文名称	适用图类型
撤销	undo	All
重做	redo	All
添加	add	All
更新	update	All
删除	remove	All
复制	copy	All
粘贴	paste	All
粘贴到这里	pasterHere	All
放大	zoomIn	All
缩小	zoomOut	All
插入主题	topic	脑图
插入子主题	subtopic	脑图
收起	fold	脑图
展开	unfold	脑图

事件监听

抛砖引玉

ggeditor 中有这么三个功能：右键菜单、标签编辑、元素浮层，它们不是 ggeditor 的核心能力，而是插件。

场景

我想做这么一个 DetailPanel（详情面板）扩展功能：选择节点、边、画布后，详情面板会展示所选元素的一些数据。

官方文档的DetailPanel（详情面板）中，虽然实现了该功能，但是代码（个人感觉）不清晰，而且 React 版本和 AntDesign 版本都没跟上，且使用的 Typescript，一时间让我摸不着头脑。

所以我去看了它们三个插件的源码，希望寻找到它们是如何监听一些事件的。

MindMap（脑图）