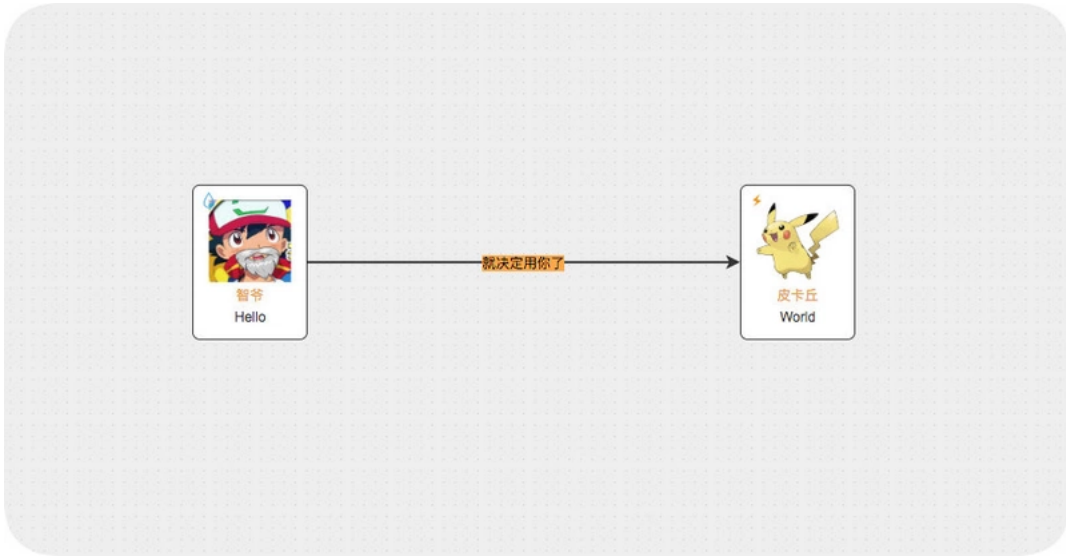


[专栏](#) / [文章详情](#)

yejinzhan RP 306
2019-03-14 发布

mxGraph 入门实例教程



在上一篇文章 [《记一次绘图框架技术选型: jsPlumb VS mxGraph》](#) 中, 提到了我为什么要去学习 mxGraph。在入门时我遇到了以下几个问题

- 官方文档偏向理论, 没能较好地结合代码进行讲解
- 虽然官方给出的例子很多, 但没有说明阅读顺序, 对刚入门的我不知道应该从哪开始阅读
- 通过搜索引擎搜索 “mxGraph教程” 没能得到太大帮助

通过自己对着官方文档死磕了一段时间并在公司项目中进行实践后, 慢慢开始掌握这个框架的使用。下面我就根据我的学习经验写一篇比较适合入门的文章。

[官方](#)列了比较多文档, 其中下面这几份是比较有用的。

- [mxGraph Tutorial](#), 这份文档主要讲述整个框架的组成
- [mxGraph User Manual – JavaScript Client](#), 这份文档对一些重要的概念进行讲解, 以及介绍一些重要的 API
- [在线实例](#), 这些实例的源码都在[这里有](#)
- [API 文档](#), 这是最重要的一份文档, 在接下来的教程我不会对接口作详细讲述, 你可以在这里对相关接口作深入了解

在看完我的文章后希望系统地学习 mxGraph 还是要去阅读这些文档的, 现在可以暂时不看。因为刚开始就堆这么多理论性的东西, 对入门没有好处。

这篇教程分为两部分, 第一部分结合我写的一些例子讲解基础知识。第二部分则利用第一部分讲解的知识开发一个小项目 [pokemon-diagram](#)。本教程会使用到 ES6 语法, 而第二部分的项目是用 Vue 写的。阅读本教程需要你掌握这两项预备知识。

引入

使用 script 引入

我们来分析一下官方的 [HelloWorld](#) 实例是怎样通过 script 标签引入 mxGraph 的

[首页](#)[问答](#)[专栏](#)[讲堂](#)[更多](#)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Hello World</title>
</head>

<body>
<div id="graphContainer"></div>
</body>

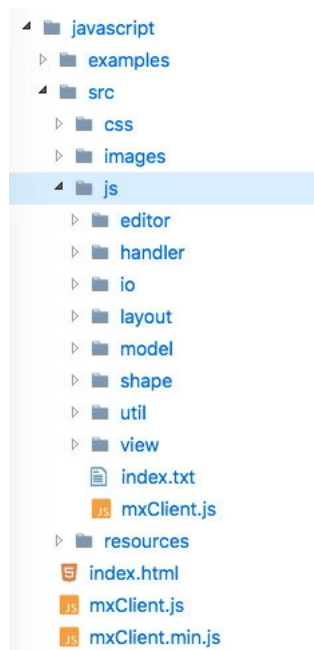
<script>
mxBasePath = '../src';
</script>

<script src="../src/js/mxClient.js"></script>
<script>
  // .....
</script>
</html>
```

首先要声明一个全局变量 `mxBasePath` 指向一个路径，然后引入 `mxGraph`。



`mxBasePath` 指向的路径作为 `mxGraph` 的静态资源路径。上图是 HelloWorld 项目的 `mxBasePath`，这些资源除了 `js` 目录，其他目录下的资源都是 `mxGraph` 运行过程中所需要的，所以要在引入 `mxGraph` 前先设置 `mxBasePath`。



再看看 `javascript` 目录下有两个 `mxClient.js` 版本。一个在 `javascript/src/js/mxClient.js`，另一个在 `javascript/mxClient.js`，后者是前者打包后的版本，所以两者是可以替换使用的。如果你的项目是使用 `script` 标签引入 `mxGraph`，可以参考[我这个库](#)。

模块化引入

模块化引入可以参考 [pokemon-diagram](#) 的这个文件 [static/mxgraph/index.js](#)

```
/** 引入 mxgraph */
// src/graph/index.js
import mx from 'mxgraph';

const mxgraph = mx({
  mxBasePath: '/static/mxgraph',
});

//fix BUG https://github.com/jgraph/mxgraph/issues/49
window['mxGraph'] = mxgraph.mxGraph;
window['mxGraphModel'] = mxgraph.mxGraphModel;
window['mxEditor'] = mxgraph.mxEditor;
window['mxGeometry'] = mxgraph.mxGeometry;
window['mxDefaultKeyHandler'] = mxgraph.mxDefaultKeyHandler;
window['mxDefaultPopupMenu'] = mxgraph.mxDefaultPopupMenu;
window['mxStylesheet'] = mxgraph.mxStylesheet;
window['mxDefaultToolbar'] = mxgraph.mxDefaultToolbar;

export default mxgraph;

/** 在其他模块中使用 */
// src/graph/Graph.js
import mxeraph from './index':
```

这里有两点需要注意的

- `mx` 方法传入的配置项 `mxBasePath` 指向的路径一定要是一个可以通过 url 访问的静态资源目录。举个例子, `pokemon-diagram` 的 [static 目录](#)是个静态资源目录, 该目录下有 `mxgraph/css/common.css` 这么个资源, 通过 `http://localhost:7777` 可以访问 `pokemon-diagram` 应用, 那么通过 `http://localhost:7777/static/mxgraph/css/common.css` 也应该是可以访问 `common.css` 才对
- 如果你是通过 `script` 标签引入 `mxGraph`, 是不需要绑定全局变量那段代码的。模块化引入要使用这段代码是因为, `mxGraph` 这个框架有些代码是通过 `window.mxXXX` 对以上属性进行访问的, 如果不做全局绑定使用起来会有点问题。这是官方一个未修复的 BUG, 详情可以查阅上面代码注释的 issue

基础知识

这部分会使用到我自己编写的一些[例子](#)。大家可以先把代码下载下来, 这些例子都是不需要使用 `node` 运行的, 直接双击打开文件在浏览器运行即可。

Cell

`Cell` 在 `mxGraph` 中可以代表 [组\(Group\)](#)、[节点\(Vertex\)](#)、[边\(Edge\)](#), `mxCell` 这个类封装了 `Cell` 的操作, 本教程不涉及到 [组](#) 的内容。下文若出现 `Cell` 字眼可以当作 [节点](#) 或 [边](#)。

事务

官方的 [HelloWorld](#) 的例子向我们展示了如何将节点插入到画布。比较引人注意的是 `beginUpdate` 与 `endUpdate` 这两个方法, 这两个方法在官方例子中出场频率非常高, 我们来了解一下他们是干嘛用的, 嗯, 真是只是了解一下就可以了, 因为官方对两个方法的描述对入门者来说真的是比较晦涩难懂, 而且我在实际开发中基本用不上这两个方法。可以等掌握这个框架基本使用后再回过头来研究。下面的描述来源这个[文档](#), 我来简单概括一下有关这两个方法的相关信息。

2.5 Vertices and Edges

To insert vertices and edges, `beginUpdate` and `endUpdate` are used to create a transaction. The `endUpdate` should always go into a finally-block to make sure it is always executed if the `beginUpdate` was executed. However, the `beginUpdate` should not be part of the try-block to make sure `endUpdate` is never executed if `beginUpdate` fails. This is required for the model to remain in a consistent state, that is, for each call to `beginUpdate` there should always be exactly one call to `endUpdate`.

The part within the try-block creates the vertices and edges for the graph. The default parent is obtained from the graph and is typically the first child of the root cell in the model, which is created automatically when using the graph model c'tor with no arguments.

```
// Gets the default parent for inserting new cells. This
// is normally the first child of the root (ie. layer 0).
var parent = graph.getDefaultParent();

// Adds cells to the model in a single step
model.beginUpdate();
try
{
    var v1 = graph.insertVertex(parent, null, 'Hello,', 20, 20, 80, 30);
    var v2 = graph.insertVertex(parent, null, 'World!', 200, 150, 80, 30);
    var e1 = graph.insertEdge(parent, null, '', v1, v2);
}
finally
{
    // Updates the display
    model.endUpdate();
}
```

The use of `beginUpdate` and `endUpdate` does not only improve the display performance, but it is also used to mark the boundaries for undoable changes when undo/redo is used.

- `beginUpdate`、`endUpdate` 用于创建一个事务，一次 `beginUpdate` 必须对应一次 `endUpdate`
- 为了保证，假如 `beginUpdate` 执行失败，`endUpdate` 永远不会被调用，`beginUpdate` 一定要放到 `try` 块之外
- 为了保证，假如 `try` 块内更新失败，`endUpdate` 也一定被调用，`beginUpdate` 一定要放到 `finally` 块
- 使用 `beginUpdate` 与 `endUpdate` 可提高更新视图性能，框架内部做撤消/重做管理也需要 `beginUpdate`、`endUpdate`

按照官方这个说明，如果我不需要撤消/重做功能，是不是可以不使用这两个方法呢。我试着把这两个方法从 HelloWorld 例子的代码中删掉，结果程序还是可以正常运行。

insertVertex

```
mxGraph.prototype.insertVertex = function(parent, id, value,
                                          x, y, width, height, style, relative) {

    // 设置 Cell 尺寸及位置信息
    var geometry = new mxGeometry(x, y, width, height);
    geometry.relative = (relative != null) ? relative : false;

    // 创建一个 Cell
    var vertex = new mxCell(value, geometry, style);
    // ...
    // 标识这个 Cell 是一个节点
    vertex.setVertex(true);
    // ...

    // 在画布上添加这个 Cell
    return this.addCell(vertex, parent);
};
```

上面是经简化后的 `insertVertex` 方法。insertVertex 做了三件事，先是设置几何信息，然后创建一个节点，最后将这个节点添加到画布。`insertEdge` 与 `insertVertex` 类似，中间过程会调用 `vertex.setEdge(true)` 将 `Cell` 标记为边。从这里我们也可以得知无论 `节点` 还是 `边` 在 `mxGraph` 中都是由 `mxCell` 类表示，只是在该类内部标识当前 `Cell` 是 `节点` 还是 `边`。

mxGeometry



首页



问答



专栏



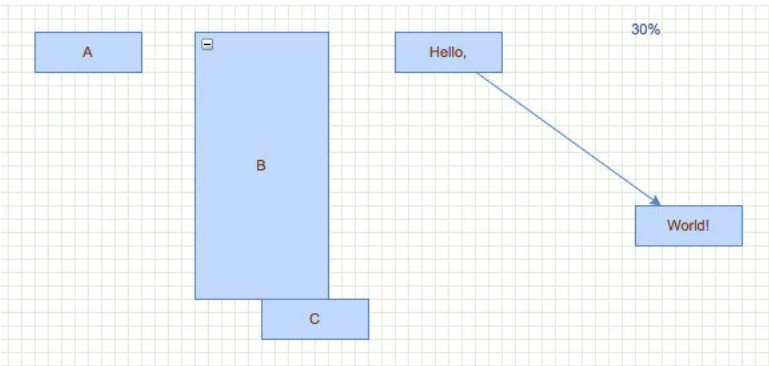
讲堂



更多

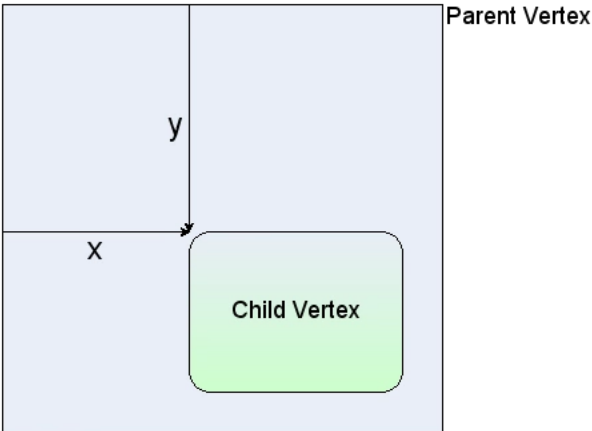
```
function mxGeometry(x,y,width,height){}
```

`mxGeometry` 类表示 `Cell` 的几何信息，宽高比较好理解，只对节点有意义，对边没意义。下面通过 [02.geometry.html](#) 这个例子说明如 `x`、`y` 的作用。

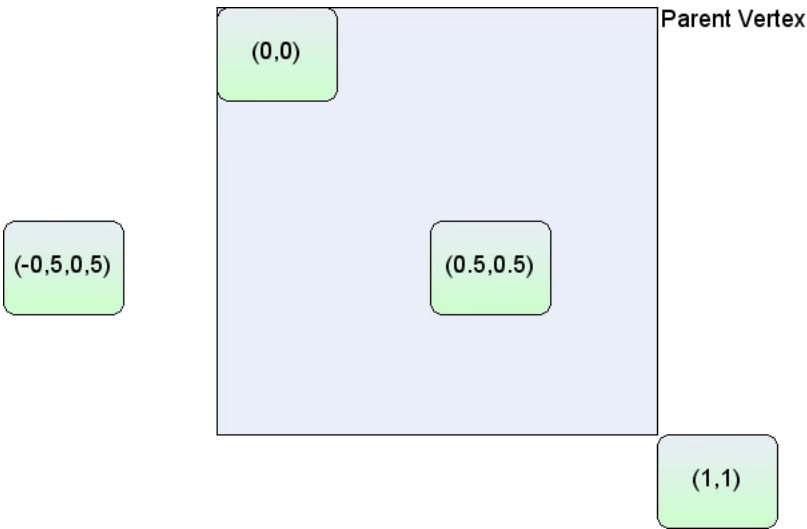


`mxGeometry` 还有一个很重要的布尔属性 `relative` ,

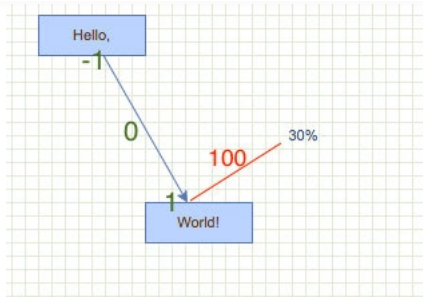
- `relative` 为 `false` 的节点，表示以画布左上角为基点进行定位，`x`、`y` 使用的是绝对单位
- 上一小节提到 `insertVertex` 内部会创建 `mxGeometry` 类。使用 `mxGraph.insertVertex` 会创建一个 `mxGeometry.relative` 为 `false` 的节点，如 A 节点



- `relative` 为 `true` 的节点，表示以父节点左上角为基点进行定位，`x`、`y` 使用的是相对单位
- 使用 `mxGraph.insertVertex` 会创建一个 `relative` 为 `false` 的节点。如果你要将一个节点添加到另一个节点中需要在该方法调用的第9个参数传入 `true` , 将 `relative` 设置为 `true` 。这时子节点使用相对坐标系，以父节点左上角作为基点，`x`、`y` 取值范围都是 `[-1,1]` 。如 C 节点相对 B 节点定位。

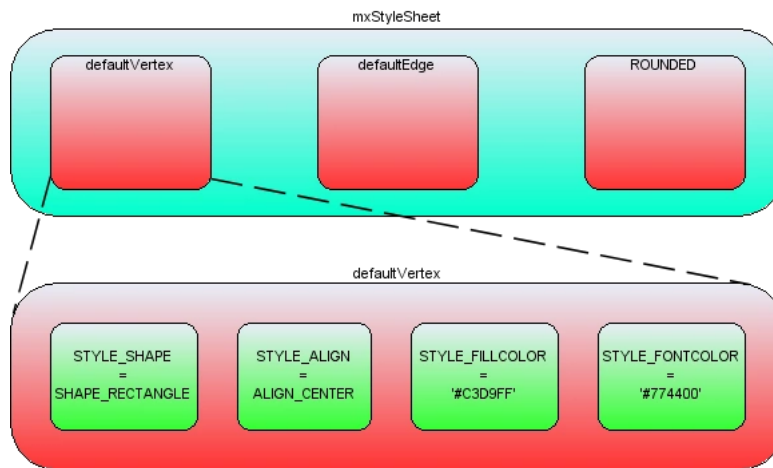


- `relative` 为 `true` 的边，`x`、`y` 用于定位 label
- 使用 `mxGraph.insertEdge` 会创建一条 `relative` 为 `true` 的边。`x`、`y` 用于定位线条上的 label，`x` 取值范围是 `[-1,1]` , `-1` 为起点，`0` 为中点，`1` 为终点。`y` 表示 label 在边的正交线上移到的距离。第三个例子能帮忙大家理解这种情况。



```
const e1 = graph.insertEdge(parent, null, '30%', v1, v2);
e1.geometry.x = 1;
e1.geometry.y = 100;
```

设置样式



由 [03.stylesheet.html](https://segmentfault.com/a/1190000018510996?utm_source=tag-newest) 这个例子我们得知 mxGraph 提供两种设置样式的方式。

第一种是设置 **全局样式**。[mxStylesheet](#) 类用于管理图形样式，通过 [graph.getStylesheet\(\)](#) 可以获取当前图形的 [mxStylesheet](#) 对象。[mxStylesheet](#) 对象的 [styles](#) 属性也是一个对象，该对象默认情况下包含两个对象 [defaultVertexStyle](#)、[defaultEdgeStyle](#)，修改这两个对象里的样式属性 **对所有线条/节点都生效**。

第二种是 **命名样式**。先创建一个样式对象，然后使用 [mxStylesheet.putCellStyle](#) 方法为 [mxStylesheet.styles](#) 添加该样式对象并命名。在添加 Cell 的时候，将样式写在参数中。格式如下

```
[stylename;|key=value;]
```

分号前可以跟命名样式名称或者一个样式的 key、value 对。

ROUNDED 是一个内置的命名样式，对节点设置有圆角效果，对边设置则边的拐弯处为圆角。

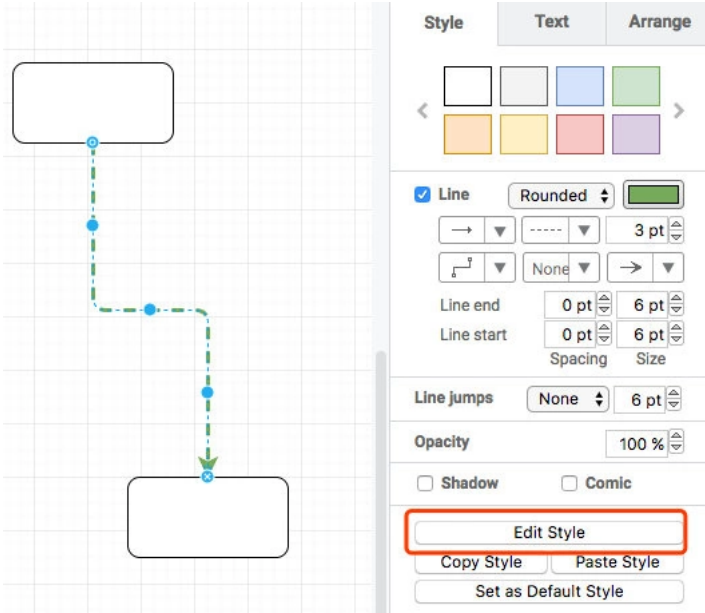
例子中设置折线有一个需要注意的地方。

```
// 设置拖拽边的过程出现折线，默认为直线
graph.connectionHandler.createEdgeState = function () {
  const edge = this.createEdge();
  return new mxCellState(graph.view, edge, graph.getCellStyle(edge));
};
```

虽然调用 [insertEdge](#) 方法时已经设置了线条为折线，但是在拖拽边过程中依然是直线。上面这段代码重写了 [createEdgeState](#) 方法，将拖动中的边样式设置成与静态时的边样式一致，都是折线。

mxGraph 所有样式在[这里](#)可以查看，打开网站后可以看到以 `STYLE_` 开头的是样式常量。但是这些样式常量并不能展示样式的效果。下面教大家一个查看样式效果的小技巧，使用 [draw.io](#) 或 [GraphEditor](#) (这两个应用都是使用 mxGraph 进行开发的) 的 `Edit Style` 功能可以查看当前 Cell 样式。

比如现在我想将边的样式设置成：折线、虚线、绿色、拐弯为圆角、粗3pt。在 Style 面板手动修改样式后，再点击 `Edit Style` 就可以看到对应的样式代码。



Edit Style:

```
edgeStyle=orthogonalEdgeStyle;
rounded=1;
orthogonalLoop=1;
jettySize=auto;
html=1;
strokeWidth=3;
fillColor=#d5e8d4;
strokeColor=#82b366;
dashed=1;
entryX=0.5;entryY=0;entryDx=0;entryDy=0;
```

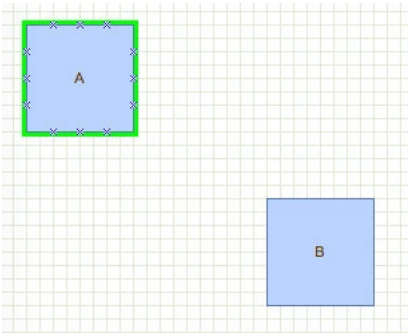
CancelApply

为了方便观察我手动格式化了样式，注意最后一行以 `entry` 或 `exit` 开头的样式代表的是边出口/入口的靶点坐标，下一小节会进行讲解。

靶点

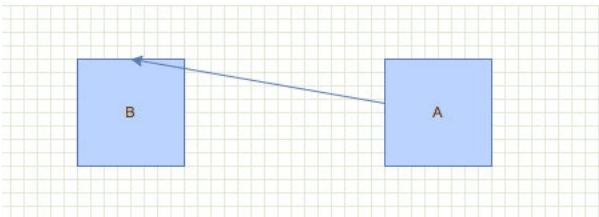
关于如何设置靶点可以参考 [04.anchors.html](#)，下面也是以这个 Demo 进行讲解两个用户操作的例子，对比不同的操作对于获取靶点信息的影响。

将鼠标悬浮中 A 节点中心，待节点高亮时连接到 B 节点的一个靶点上



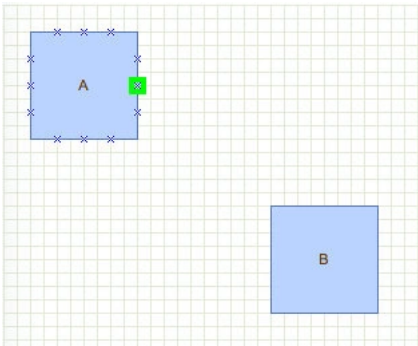


然后将 A 节点拖拽到 B 节点右边

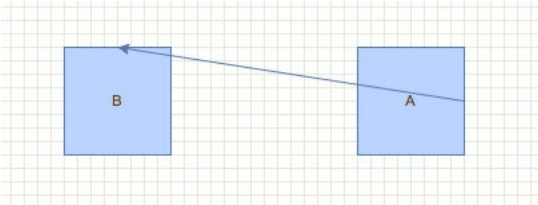


可以看到如果从图形中心拖出线条，这时边的出口值 `exit` 为空，只有入口值 `entry`。如果拖动节点 mxGraph 会智能地调整线条出口方向。如节点 A 的连接靶点原来是在右边，节点拖动到节点 B 右边后靶点也跟着发生了变化，跑到了左边，而节点 B 的连接靶点一直没变。

这次将鼠标悬浮到 A 节点的一个靶点，待靶点高亮时连接到 B 节点的一个靶点上



然后将 A 节点拖拽到 B 节点右边



可以看到这次所有值都有了，连接后拖动节点 A，连接靶点的位置也固定不变，mxGraph 不像第一个例子一样调整连接靶点位置，之所以产生这样的差异是因为第

面向对象编程

mxGraph 框架是使用面向对象的方式进行编写的，该框架所有类带 mx 前缀。在接下来的例子你会看到很多这种形式的方法 **重写(Overwrite)**。

```
const oldBar = mxFoo.prototype.bar;
mxFoo.prototype.bar = function (...args)=> {
  // .....
  oldBar.apply(this,args);
  // .....
};
```

节点组合

这一小节通过 [05.consistent.html](#) 这个例子，讲解节点组合需要注意的地方。

组合节点后默认情况下，父节点是可折叠的，要关闭折叠功能需要将 **foldingEnabled** 设为 **false**。

```
graph.foldingEnabled = false;
```

如果希望在改变父节点尺寸时，子节点与父节点等比例缩放，需要开启 **recursiveResize**。

```
graph.recursiveResize = true;
```

下面是这个例子最重要的两段代码。

```
/**
 * Redirects start drag to parent.
 */
const getInitialCellForEvent = mxGraphHandler.prototype.getInitialCellForEvent;
mxGraphHandler.prototype.getInitialCellForEvent = function (me) {
  let cell = getInitialCellForEvent.apply(this, arguments);
  if (this.graph.isPart(cell)) {
    cell = this.graph.getModel().getParent(cell);
  }
  return cell;
};

// Redirects selection to parent
graph.selectCellForEvent = function (cell) {
  if (this.isPart(cell)) {
    mxGraph.prototype.selectCellForEvent.call(this, this.model.getParent(cell));
    return;
  }

  mxGraph.prototype.selectCellForEvent.apply(this, arguments);
};
```

这两个方法 **重写(Overwrite)** 了原方法，思路都是判断如果该节点是子节点则替换成父节点去执行剩下的逻辑。

getInitialCellForEvent 在鼠标按下(mousedown事件，不是click事件)时触发，如果注释掉这段代码，不使用父节点替换，当发生拖拽时子节点会被单独拖拽，不会与父节点联动。使用父节点替换后，原本子节点应该被拖拽，现在变成了父节点被拖拽，实现联动效果。

selectCellForEvent 其实是 **getInitialCellForEvent** 内部调用的一个方法。这个方法的作用是将 cell 设置为 **selectionCell**，设置后可通过 **mxGraph.getSelectionCell** 可获取得该节点。与 **getInitialCellForEvent** 同理，如果不使用父节点替换，则 **mxGraph.getSelectionCell** 获取到的会是子节点。项目实战我们会使用到 **mxGraph.getSelectionCell** 这个接口。

项目实战



首页



问答



专栏



讲堂



更多

这部分我主要挑一些这个[项目](#)比较重要的点进行讲解。

写一个节点组合

下面以项目的这个节点为例，讲解如何组合节点



```
const insertVertex = (dom) => {
  // ...
  const nodeRootVertex = new mxCell('鼠标双击输入', new mxGeometry(0, 0, 100, 135), `node;image=${src}`);
  nodeRootVertex.vertex = true;
  // ...

  const title = dom.getAttribute('alt');
  const titleVertex = graph.insertVertex(nodeRootVertex, null, title,
    0.1, 0.65, 80, 16,
    `normalType=1;whiteSpace=wrap;strokeColor=none;fillColor=none;fontColor=#e6a23c`,
    true);
  titleVertex.setConnectable(false);

  const normalTypeVertex = graph.insertVertex(nodeRootVertex, null, null,
    0.05, 0.05, 19, 14,
    `normalType;constituent=1;fillColor=none;image=/static/images/normal-type/forest.png`,
    true);
  normalTypeVertex.setConnectable(false);
  // .....
};
```

单单 `nodeRootVertex` 就是长这个样子。通过设置自定义的 `node` 样式(见 [Graph](#) 类 `_putVertexStyle` 方法)与 `image` 属性设置图片路径配合完成。



因为默认情况下一个节点只能有一个文本区和一个图片区，要增加额外的文本和图片就需要组合节点。在 `nodeRootVertex` 上加上 `titleVertex` 文本节点和 `normalTypeVertex` 图片节点，最终达到这个效果。



有时需要为不同子节点设置不同的鼠标悬浮图标，如本项目鼠标悬浮到 `normalTypeVertex` 时鼠标变为手形，参考 `AppCanvas.vue` 的 `setCursor` 方法，重写 `mxGraph.prototype.getCursorForCell` 可以实现这个功能。

```
const setCursor = () => {
  const oldGetCursorForCell = mxGraph.prototype.getCursorForCell;
  graph.getCursorForCell = function (...args) {
    const [cell] = args;
    return cell.style.includes('normalType') ?
      'pointer' :
      oldGetCursorForCell.apply(this, args);
  };
};
```

编辑内容

下面这段代码是编辑内容比较常用的设置

```
// 编辑时按回车键不换行，而是完成输入
this.setEnterStopsCellEditing(true);
// 编辑时按 escape 后完成输入
mxCellEditor.prototype.escapeCancelsEditing = false;
// 失焦时完成输入
mxCellEditor.prototype.blurEnabled = true;
```

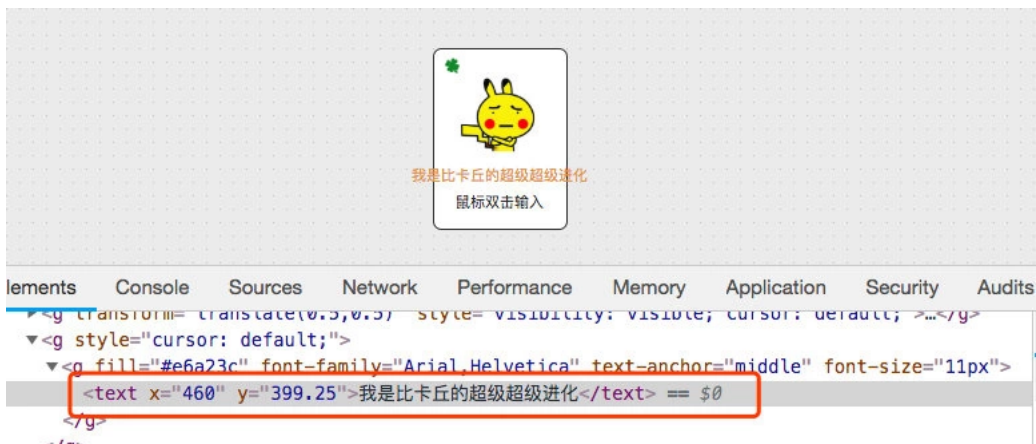
默认情况下输入内容时如果按回车键内容会换行，但有些场景有禁止换行的需求，希望回车后完成输入，通过 `graph.setEnterStopsCellEditing(true)` 设置可以满足需求。

重点说说 `mxCellEditor.prototype.blurEnabled` 这个属性，默认情况下如果用户在输入内容时鼠标点击了画布之外的不可聚焦区域(div、section、article等)，节点内的编辑器是不会失焦的，这导致了 `LABEL_CHANGED` 事件不会被触发。但在实际项目开发中一般我们会期望，如果用户在输入内容时鼠标点击了画布之外的地方就应该算作完成一次输入，然后通过被触发的 `LABEL_CHANGED` 事件将修改后的内容同步到服务端。通过 `mxCellEditor.prototype.blurEnabled = true` 这行代码设置可以满足我们的需求。

可换行的 label

```
const titleVertex = graph.insertVertex(nodeRootVertex, null, title,
  0.1, 0.65, 80, 16,
  'constituent=1;whiteSpace=wrap;strokeColor=none;fillColor=none;fontColor=#e6a23c',
  true);
```

对于非输入的文本内容，默认情况下即便文本超出容器宽度也是不会换行的。我们项目中宽度为 80 的 `titleVertex` 正是这样一个例子。



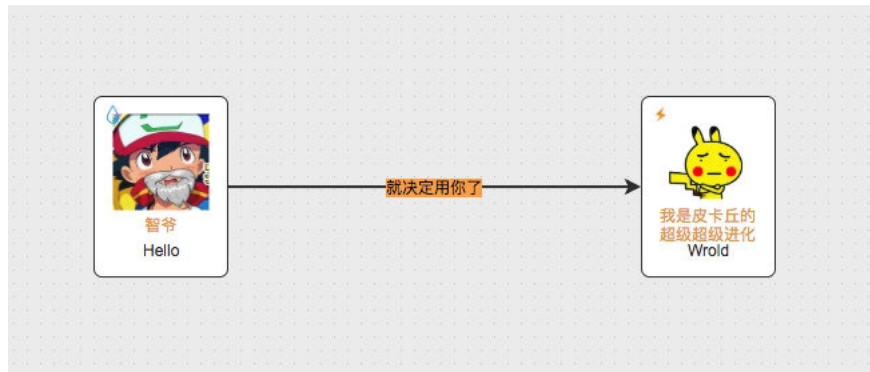
要设置换行需要做两件事，第一是通过这行代码 `mxGraph.setHtmlLabels(true)`，使用 html 渲染文本(mxGraph 默认使用 svg 的 text 标签渲染文本)。第二是像上面的 `titleVertex` 的样式设置一样，添加一句 `whiteSpace=wrap`。



Model

现在介绍一下 Model 这个概念，Model 是当前图形的数据结构化表示。[mxGraphModel](#) 封装了 Model 的相关操作。

你可以启动项目，画一个这样的图，然后点击输出XML。为了保的 xml 与下面的一致，需要先拖出智爷，再拖出超级皮卡丘，最后连接边。



控制台应该输出这样一份 xml

```
<mxGraphModel>
  <root>
    <mxCell id="0"/>
    <mxCell id="1" parent="0"/>
    <mxCell id="4" value="Hello" style="node;image=/static/images/ele/ele-005.png" vertex="1" data="{&quot;id&quot;:1,&quot;element&quot;:{
      <mxGeometry x="380" y="230" width="100" height="135" as="geometry"/>
    }"/>
    </mxCell>
    .....
  </root>
</mxGraphModel>
```

每一个 mxCell 节点都有 parent 属性指向父节点。我们对 value="Hello" 这个 mxCell 节点手动格式化。

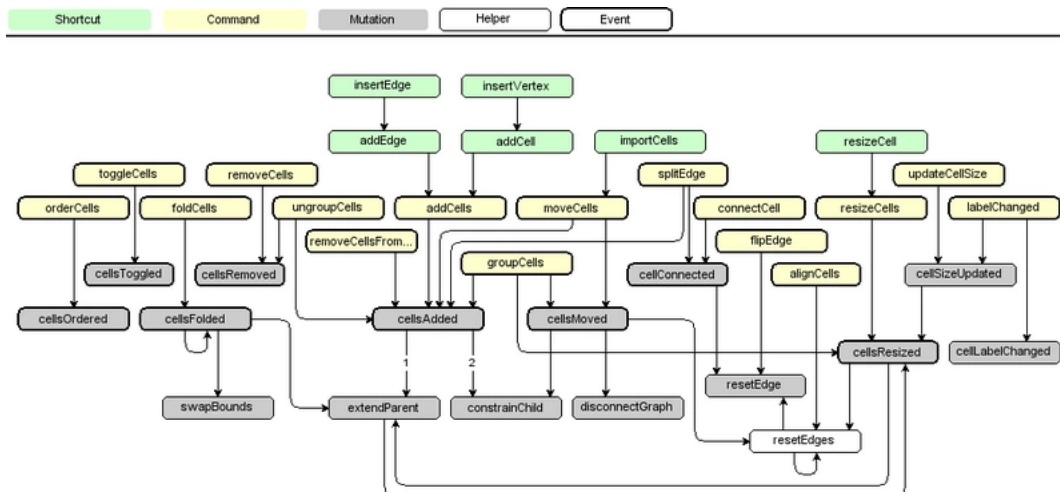
```
value="Hello"
style="node;image=/static/images/ele/ele-005.png"
vertex="1"
data="{&quot;id&quot;:1,&quot;element&quot;:{&quot;id&quot;:1,&quot;icon&quot;:&quot;ele-005.png&quot;,&quot;title&quot;:&quot;智爷&quot;}"
parent="1">
<mxGeometry
  x="380"
  y="230"
  width="100"
  height="135" as="geometry"/>
</mxCell>
```

data 值是原对象经 `JSON.stringify` 得到的，经转义后就变成了上面的样子。控制台还打印了一个 `mxGraphModel` 对象，对比上面的 xml 与 下图的节点对象，可以发现它们只是同一个 Model 的不同表现形式，xml 正是将 `mxGraph.model` 格式化而成的。

```
▼ mxGraphModel {currentEdit: mxUndoableEdit, root: mxCell, nextId: 13, cells: {...}, updateLevel: 0, ...}
  ▼ cells:
    ► 0: mxCell {...}
    ► 1: mxCell {...}
    ► 2: mxCell {value: "智爷", geometry: mxGeometry, style: "constituent=1;whiteSpace=wrap;strokeColor=non
    ► 3: mxCell {value: null, geometry: mxGeometry, style: "normalType;constituent=1;fillColor=none;image=
    ▼ 4: mxCell
      children: (...)
      clone: (...)
      cloneValue: (...)
      collapsed: (...)
      connectable: (...)
      ▼ data: Object
        ► element: Object
          id: 1
          normalType: "water.png"
        ► __ob__: Observer {value: {...}, dep: Dep, vmCount: 0}
        ► get element: f reactiveGetter()
        ► set element: f reactiveSetter(newVal)
        ► get id: f reactiveGetter()
        ► set id: f reactiveSetter(newVal)
        ► get normalType: f reactiveGetter()
        ► set normalType: f reactiveSetter(newVal)
        ► __proto__: Object
      edge: false
      ► edges: [mxCell]
      geometry: (...)
```

事件

本项目监听事件写在 `AppCanvas.vue` 的 `_listenEvent` 方法，可以在这个方法了解一些常用的事件。下图来自 `mxGraph` 类的方法调用依赖图，我们可以从这里看出整个框架的事件流动。



本项目的 `_listenEvent` 方法用到两个事件监听对象。

- `mxGraph` 继承自 `mxEventSource`，使用父类的 `addListener` 方法可以将自身当作一个事件中心进行订阅/广播事件。
- `mxGraph.getSelectionModel()` 返回一个 `mxGraphSelectionModel` 对象，这个对象也是继承自 `mxEventSource` 有 `mxEvent.UNDO`、`mxEvent.CHANGE` 两个事件，通过监听 `mxEvent.CHANGE` 事件可以获取当前被选中的 `Cell`。

ADD_CELLS 与 CELLS_ADD 的区别

<code>mxEvent.ADD_CELLS</code>	Fires between begin- and endUpdate in <code>addCells</code> .
<code>mxEvent.CELLS_ADDED</code>	Fires between begin- and endUpdate in <code>cellsAdded</code> .

`mxGraph` 类有很多 `XXX_CELLS`、`CELLS_XXXED` 这种形式的事件，这部分我还没弄懂，下面仅以添加事件为例探讨这两类事件的区别。

- 添加 `Cell` 的时候会触发两个事件 `ADD_CELLS`、`CELLS_ADDED`，先触发 `CELLS_ADDED` 后触发 `ADD_CELLS`。
- `ADD_CELLS` 在 `addCells` 方法中触发，而 `CELLS_ADDED` 在 `cellsAdded` 方法中触发。而对于 `addCells` 与 `cellsAdded` 官方文档的说明并不能体现出两者的区别，再深究下去就要查阅源码了。按经验而言后触发的事件会携带更多的信息，所以平时开发我会监听 `ADD_CELLS` 事件。`MOVE_CELLS`、`CELLS_MOVED`、`REMOVE_CELLS`、`CELLS_REMOVED` 等事件与此类似。

监听 Cell 添加事件

从上面的方法调用依赖图中我们可以看到，`insertVertex`、`insertEdge` 最终都被当作 `Cell` 处理，在后续触发的事件也没有对 `节点/边` 进行区分，而是统一当作 `Cell` 事件。所以对于一个 `Cell` 添加事件，需要自己区别是添加了节点还是添加了边。

```
graph.addListener(mxEvent.CELLS_ADDED, (sender, evt) => {
  const cell = evt.properties.cells[0];
  if (graph.isPart(cell)) {
    return;
  }

  if (cell.vertex) {
    this.$message.info('添加了一个节点');
  } else if (cell.edge) {
    this.$message.info('添加了一条线');
  }
});
```

还有就是对于子节点添加到父节点的情况(如本项目将 `titleVertex`、`normalTypeVertex` 添加到 `nodeRootVertex`)也是会触发 `Cell` 添加事件的。通常对于这些子节点不作处理，可以像 [05.consistent.html](#) 一样用一个 `isPart` 判断过滤掉。

自定义事件

上面提到过 `mxGraph` 继承自 `mxEventSource`，调用父类的 `fireEvent` 可触发自定义事件。下面是一个简单的例子

```
mxGraph.addListener('自定义事件A', ()=>{
  // do something .....
});
// 触发自定义事件
mxGraph.fireEvent(new mxEventObject('自定义事件A'));
```

在本项目 `Graph` 类的 `_configCustomEvent` 方法我也实现了两个自定义事件。当边开始拖动时会触发 `EDGE_START_MOVE` 事件，当节点开始拖动时会触发 `VERTEX_START_MOVE` 事件。

导出图片

`mxGraph` 导出图片的思路是先在前端导出图形的 `xml` 及计算图形的宽高，然后将 `xml`、宽、高，这三项数据发送给服务端，服务端也使用 `mxGraph` 提供的 API 将 `xml` 转换成图片。服务端如果是使用 Java 可以参考官方这个[例子](#)，下面主要介绍前端需要做的工作。


[首页](#)

[问答](#)

[专栏](#)

[讲堂](#)

[更多](#)


```
// ...
var xmlCanvas = new mxXmlCanvas2D(root);
var imgExport = new mxImageExport();
imgExport.drawState(graph.getView().getState(graph.model.root), xmlCanvas);

var bounds = graph.getGraphBounds();
var w = Math.ceil(bounds.x + bounds.width);
var h = Math.ceil(bounds.y + bounds.height);

var xml = mxUtils.getXml(root);
// ...
```

但这段代码会将整块画布截图，而不是以最左上角的元素及最右下角的元素作为边界截图。如果你有以元素作为边界的需求，则需要调用 [xmlCanvas.translate](#) 调整截图边界。

```
//.....
var xmlCanvas = new mxXmlCanvas2D(root);
xmlCanvas.translate(
    Math.floor((border / scale - bounds.x) / scale),
    Math.floor((border / scale - bounds.y) / scale),
);
//.....
```

完整截图代码可以参考本项目 [Graph](#) 类的 `exportPicXML` 方法。

如果节点像我的项目一样使用到图片，而导出出来的图片的节点没有图片。可以从两个方向排查问题，先检查发送的 xml 里的图片路径是否是可访问的，如下面是项目“导出图片”功能打印的 xml 里的一个图片标签。

```
<image x="484" y="123" w="72" h="72" src="http://localhost:7777/static/images/ele/ele-005.png" aspect="0" flipH="0" flipV="0"/>
```

要保证 `http://localhost:7777/static/images/ele/ele-005.png` 是可访问的。如果图片路径没问题再检查一下使用的图片格式，本来我在公司项目中节点内使用的图片是 svg 格式，导出图片失败，可能是 mxGraph 不支持这个格式，后来换成 png 之后问题就解决了。

还有就是如果导出的图片里的节点的某些颜色跟设置的有差异，那可能是设置样式时写了3位数的颜色像 `#fff`，颜色一定要使用完整的6位，否则导出图片会有问题。

参考

- [mxGraph Tutorial](#)
- [mxGraph User Manual – JavaScript Client](#)
- [mxGraph API Specification](#)
- [mxGraph Javascript Examples](#)



赞 | 21

收藏 | 16



学生服务器体验套餐10元/月
• 1核2G • 1M宽带 • 50GB存储

你可能感兴趣的



首页



问答



专栏



讲堂



更多

- Ember.js入门教程、博文汇总

ember_teach

javascript

ember.js
- 【译】Flask-Admin中文入门教程

digwtx

flask-admin

flask
- 我看完227篇文章，总结出一份Flutter入门教程

掘金

前端

ios

android

flutter
- vue - 教程 - 入门

limichange

vue.js
- 个人文章分类整理

samsara0511

apache

vue.js

html

javascript

css
- vue入门文章

stray

css
- Gulp入门教程

墨鱼

gulp

javascript

28 条评论

默认排序时间排序

cheesemp

· 3月15日

建议开头先简单介绍下什么东西...

👍 赞

回复

SHERlocked93

· 3月15日

不错的文章，感谢分享 ~

👍 赞

回复

乐不起

· 3月15日

mxgraph功能特别齐全，就是和webpack有兼容问题.....

👍 赞

回复

跟webpack兼容有什么问题，我暂时还没发现

— yejinzhan 作者 · 3月15日

添加回复

李狗蛋

· 3月27日

楼主大大可以介绍下该如何学习mxgraph吗。我把你的这篇文章看完之后，文章中的例子都可以理解，但是对着实际代码的时候还是有种无从下手的感觉。对于没见过
的mxgraph内置方法还是较难理解他的作用。

👍 赞

回复

最好是以项目为导向来学，mxGraph 是一个大型的库，对于初始学不可能把文档从头到尾看一遍。pokemon-diagram 是我学习 mxGraph 的时候写的，不是因为做这个教
程而写的，只是写这个教程的时候把这个项目装饰得好看点而已。我学 mxGraph 就是为了解决公司项目的画图难题，于是以公司项目需求作为参考做了 pokemon-
diagram。遇到什么需求就在官方的 demo 里找，看看哪个 demo 符合，就把代码加到项目里，遇到不懂的 api 就查查文档。比如拖拽元素到画布这个小功能点，就在
demo 里面找哪个 demo 实现了这个功能。慢慢你就会熟悉官方那些 demo，当你对官方那些 demo 越来越熟就入门了

— yejinzhan 作者 · 3月27日

添加回复

sunshine

· 4月9日

感谢分享，雪中送碳啊，官方文档全英文，一点也不想看.....

👍 赞

回复

dolphinfine

· 4月28日

很详细的教程。感谢

👍 赞

回复

木木木木木dian

· 5月13日


你好，请问下 exportPicXML 有服务端是 nodejs 的例子吗

👍 赞

回复

服务端只支持 php、.Net、Java

— yejinzhan 作者 · 5月14日



吕杭 · 5月29日

楼主大大，这个是免费的么，需要商业授权么

👍 赞

回复

这个库用的是跟echart同一个开源协议，可以放心使用

— yejinzhan 作者 · 5月31日

添加回复



donhanttt · 6月18日

感谢分享，请问模块化引入时，mx 方法传入的配置项 mxBasePath 指向的路径一定要是一个可以通过 url 访问的静态资源目录，这个需要webpack配置什么吗？

👍 赞

回复

webpack 需要做静态资源目录配置的，可以参考这里 <https://github.com/jinzhan/...>，我是用 vue 脚手架生成的项目

— yejinzhan 作者 · 6月25日


@yejinzhan 链接404了，这个问题我已经解决了。感觉mxGraph入门真难，不知能否加一下您的微信？

— donhanttt · 6月27日

那就好，有任何问题都可以在这里提问，我会及时回答

— yejinzhan 作者 · 6月27日

添加回复



ldsis · 7月2日

请教下楼主，我遇到个问题，我加载一个数据量很大的xml，导致加载速度十分缓慢。有什么方法或者思路能提升加载速度吗？


👍 赞

回复

抱歉我还遇到这种问题，用公司项目试过画很多个节点的图，但也没有严重的加载慢问题。为什么出现数据量很大的 xml，是业务的问题么？

— yejinzhan 作者 · 7月3日

添加回复



liuqiyu · 7月5日

请问如何绑定数据。

👍 赞

回复



paperplane · 3 天前

楼主，你好，有个问题咨询下，在拖动cell的过程中，节点只能显示形状，看不到里面的文字和图片。如何展示拖动过程中的预览图，mxCellStatePreview的功能似乎就是这个，但是我试了，好像没有效果。。

👍 赞

回复

没有这个功能，官方也不打算做 <https://github.com/jgraph/mxg...>

— yejinzhan 作者 · 3 天前

给你点赞，楼主

— paperplane · 2 天前

楼主，还有个问题：父节点里添加子节点，子节点的定位为相对定位。子节点是用来当做出入口的（相当于父节点的锚点），这个时候如果多个父节点已经连接了，在重新连接时会连不上去。但是如果把子节点的相对定位去掉就可以重连了

— paperplane · 2 天前

添加回复 | 显示更多




TwiSted · 3 天前


楼主你好，请教一个问题，如何使用xml生成toolbar工具栏呢？我看了官方的diagrameditor这个例子，但是我自己试的时候却出不来，好像是因为mxUtils.load('xml').getDocumentElement()获取节点的时候出错了，这是不是还需要启动什么服务啊？

👍 赞

回复

 首页

 问答

 专栏

 讲堂

 更多

你是用 script 的方式引入 mxGraph 还是 import 的方式引入

— yejinzhan 作者 · 3 天前

因为是在vue项目中使用的 所以是import引入的 我按照官方例子diagrameditor.html写了之后页面中并没有出现toolbar 然后我在例子中也没找到生成toolbar的相关代码

— TwiSted · 2 天前

diagrameditor 例子代码太多，我建议你看 uiconfig.html 的例子<https://github.com/jgraph/mxg...>
然后你需要保证已经按文本「模块化引入」小节给 window 绑定了些属性，mxUtils.load 其实是发了一个 ajax 请求，你可以在调试面板查看网络请求，看看 xml 路径有没有找到对应资源。mxUtils.load 源码并不复杂，你直接 debugger 进去看看

— yejinzhan 作者 · 2 天前

添加回复



文明社会，理性评论

发布评论



腾讯云

学生服务器体验套餐10元/月

• 1核2G • 1M宽带 • 50GB存储



Copyright © 2011-2019 SegmentFault. 当前呈现版本 19.02.27
浙ICP备 15005796号-2 浙公网安备 33010602002000号 杭州堆栈科技有限公司版权所有

CDN 存储服务由 又拍云 赞助提供

移动版 桌面版



首页



问答



专栏



讲堂



更多