**GoJS图表组件简介**

**GoJS**是一个JavaScript库，可以让您轻松地在现代网络浏览器中创建交互式图表。 **GoJS**支持图形模板和图形对象属性与模型数据的数据绑定。您只需要保存和恢复模型，其中包含简单的JavaScript对象，其中包含您的应用需要的任何属性。许多预定义的工具和命令实现大多数图表需要的标准行为。外观和行为的定制主要是设置属性的问题。

一个简单的GoJS图

以下代码定义了一个节点模板和模型数据，该模型和模型数据生成了一个小图，其中包含少量的节点和链接。

//为了简洁。有关更多信息，请参阅“建筑零件”简介页面

var $ = go.GraphObject.make;

//节点模板描述如何构建每个节点

diagram.nodeTemplate =

$（go.Node，“Auto”，// Shape自动适应TextBlock

$（go.Shape，“RoundedRectangle”，//使用这种形状的形状

//将Shape.fill绑定到Node.data.color

new go.Binding（“fill”，“color”）），

$（go.TextBlock，

{margin：3}，//文本周围有一些空间

//将TextBlock.text绑定到Node.data.key

new go.Binding（“text”，“key”））

）;

//模型只保存描述图的基本信息

diagram.model = new go.GraphLinksModel（

[// JavaScript对象的JavaScript数组，每个节点一个;

//这个应用专门添加了“color”属性

{key：“Alpha”，color：“lightblue”}，

{key：“Beta”，color：“orange”}，

{key：“Gamma”，颜色：“lightgreen”}，

{key：“Delta”，color：“pink”}

]

[// JavaScript对象的JavaScript数组，每个链接一个

{from：“Alpha”，to：“Beta”}，

{from：“Alpha”，to：“Gamma”}，

{from：“Beta”，至：“Beta”}，

{from：“Gamma”，to：“Delta”}，

{from：“Delta”，to：“Alpha”}

]）;

diagram.initialContentAlignment = go.Spot.Center;

//启用Ctrl-Z撤消，Ctrl-Y重做

diagram.undoManager.isEnabled = true;

这将创建以下图表：

您可以通过多种方式与该图进行交互：

* 您可以通过点击它来选择零件。所选节点用包围节点的蓝色矩形的“ 装饰”突出显示。所选链接以链接的路径后面的蓝线突出显示。
* 可以一次选择多个部分。单击添加到选择中时，按住Shift键。当单击以切换是否选择该部分时，按住Control键。
* 多选择的另一种方式是在背景中的一点（而不是部分）上鼠标，等待片刻，然后拖动一个框。选择鼠标出现时在框中的部件。Shift和Control修饰符也可以工作。
* Ctrl-A选择图中的所有部分。
* 通过选择和拖动来移动一个或多个节点。
* 复制选择的部分可以复制/粘贴（Ctrl-C / Ctrl-V）或使用Ctrl-mouse拖动。
* 使用删除键删除所选零件。
* 如果滚动条可见，或者整个零件集合小于图表的可视区域（“视口”），则可以在后台（而不是部件）上以鼠标方式平移图表，如果拖动时不拖动等候。
* 使用鼠标滚轮向上和向下滚动，Shift - 鼠标滚轮向左和向右滚动。Ctrl-mouse-wheel放大缩小。

您还可以通过手指在触摸设备上平移，捏缩放，选择，复制，移动，删除，撤消和重做。可以从键盘调用的大多数命令可以从默认的上下文菜单中调用，通过按压您的手指并将其静止一会儿即可获得。

文档中所有示例的独特之处在于它们都是“实时的” - 没有截图！它们是由源代码实现的实际图。您可以与他们互动 - 甚至可以显示动画。

如果您想了解更多**GoJS**可以做的[示例](http://gojs.net/latest/samples/index.html)，请参阅[GoJS Samples目录](http://gojs.net/latest/samples/index.html)。为了更方便地搜索JavaScript代码和文档，或通过修改示例进行实验，您可以以各种方式安装**GoJS**套件：

* 从[下载](http://gojs.net/latest/doc/download.html)下载ZIP文件。
* 从[GitHub的GoJS](https://github.com/NorthwoodsSoftware/GoJS)下载我们。
* 安装GoJS使用npm install gojs。

GoJS概念

图表由部分 s组成：可以通过链接连接并可以组合在一起成组的节点。所有这些部分都以层 s 聚集在一起，并通过布局布置。

每个图都有一个模型，用于保存和解释您的应用程序数据，以确定节点到节点的链接关系和组成员关系。大多数部件都是数据绑定到您的应用程序数据。该图自动为模型的Model.nodeDataArray中的每个数据项创建一个节点或组， 并为模型的GraphLinksModel.linkDataArray中的每个数据项创建一个链接。您可以为每个数据对象添加所需的任何属性，但每种类型的模型只需要几个属性。

每个节点或链接通常由声明其外观和行为的模板定义。每个模板由面板号第GraphObject S，从而如TextBlock的 S或形状秒。所有部件都有默认模板，但几乎所有应用程序都将指定自定义模板，以达到所需的外观和行为。GraphObject属性与模型数据属性的数据绑定使数据的每个节点或链接都是唯一的。

该节点可以被手动地定位（交互方式或编程），或者可以由自动排列 Diagram.layout以及由每个Group.layout。节点由左上角点（GraphObject.position）或节点中的程序员定义的点（Part.location和Part.locationSpot）定位。

工具处理鼠标和键盘事件。每个图都有许多工具执行交互式任务，例如选择部件或拖动它们或在两个节点之间绘制新的链接。该ToolManager确定哪些工具应该运行，根据鼠标事件和目前的情况。

每个图还有一个CommandHandler实现各种命令，如删除或复制。CommandHandler在ToolManager运行时解释键盘事件，例如control-Z。

该图提供了滚动图形部分并放大或缩小的能力。该图还包含所有层，其又包含所有部分（节点和链接）。这些部分又由可能嵌套的文本，形状和图像的面板组成。内存中的JavaScript对象的这种层次结构形成了图中可能绘制的所有内容的“视觉树”。

在概述类允许用户看到整个模型和控制一下它的一部分，该图显示。的调色板类认为，用户可以拖动和放下到的图的部分。

您可以在图表中选择一个或多个零件。模板实现可以在选择节点或链接的外观时改变。该图还可以添加装饰以指示选择并支持诸如调整节点大小或重新连接链接的工具。装饰也是如何实现工具提示和上下文菜单。

对图表，GraphObject，模型或模型数据状态的所有程序化更改应在每个用户操作的单个事务中执行，以确保更新正确执行并支持撤消/重做。所有预定义的工具和命令执行事务，因此如果启用了UndoManager，每个用户操作都将自动撤消。 图表上的DiagramEvent和Diagrams和GraphObjects上的事件处理程序都被记录在文件中，无论是在事务中引发，还是需要进行事务才能更改模型或图表。

创建图

**GoJS**不依赖于任何JavaScript库或框架，因此您应该可以在任何环境中使用它。然而，它要求环境支持现代的HTML和JavaScript。

加载GoJS

在执行任何JavaScript代码构建图之前，您需要加载**GoJS**库“go.js”。当您包含库时，“ go”JavaScript对象将保存所有的**GoJS**类型。在开发期间，我们建议您加载“go-debug.js”而不是“go.js”，以获得更多的运行时错误检查和调试能力。

我们建议您声明您的网页支持现代HTML：

<！DOCTYPE html> <！ - 声明标准模式。 - >

<HTML>

<HEAD>

。。。

<！ - 包括GoJS库。 - >

<script src =“go.js”> </ script>

如果您使用[RequireJS](http://requirejs.org/)，**GoJS**支持UMD模块定义。有关[示例](http://gojs.net/latest/samples/require.html)，请参阅[需求示例](http://gojs.net/latest/samples/require.html)。此外，扩展类的模块化版本现在可以在../extensionsTS/其中将扩展类转换为TypeScript并编译成.js可以import编辑的文件。或required。

在ES6（ECMAScript 2015）或TypeScript代码中，只需导入“go.js”库：

进口\*从“走”走;

或者，如果依赖于您的npm环境：

进口\*从“gojs”;

在Div元素中托管GoJS

每个图表必须由HTML Div元素托管。 **GoJS**将管理该Div元素的内容，但您可以按照任何HTML元素的位置，大小和风格进行Div。该图将添加一个Canvas元素到该图将绘制的该Div元素 - 这是用户实际看到的。Canvas元素的大小自动调整为与Div元素大小相同。

<BODY>

。。。

<！ - 图表的DIV需要一个明确的大小，否则我们看不到任何东西。

在这种情况下，我们还添加一个边框来帮助查看边缘。 - >

<div id =“myDiagramDiv”style =“border：solid 1px blue; width：400px; height：150px”> </ div>

然后，您可以在JavaScript中创建Diagram，并引用该Div元素。通过构建纯JavaScript对象并将其添加到图的模型来构建图。请注意，JavaScript代码中的所有**GoJS**类型（如Diagram）的引用都以“ go.” 为前缀。

<！ - 使用JavaScript在DIV元素中创建图。 - >

<！ - “go”对象是包含所有GoJS类型的“命名空间”。 - >

<SCRIPT>

var diagram = new go.Diagram（“myDiagramDiv”）;

diagram.model = new go.GraphLinksModel（

[{key：“Hello”}，//两个节点数据，在一个数组中

{key：“世界！” }]，

[{from：“Hello”，to：“World！”}] //一个链接数据，在一个数组中

）;

</ SCRIPT>

这完成了“Hello World！”的实现。你上面看到的活动图。

开发图

当出现问题时，**GoJS**输出错误或警告信息。使用**GoJS**进行开发时，请务必查看浏览器的开发者控制台以获取信息。库的“go-debug.js”版本包含额外的类型检查和错误检查代码，应在开发过程中使用。“go.js”版本的错误检查较少，但结果更快，应在生产过程中使用。

您的JavaScript代码只能使用[API](http://gojs.net/latest/api/index.html)中记录的属性和方法。该**GoJS**库“精缩”，所以如果你看一个实例**GoJS**在调试器类，你会看到许多的一个或两个字母的属性名称。所有这些都是您不应使用的内部名称。在当前时间，Point，Rect，Spot和LayoutVertex上唯一的一个字母属性名称为“x”和“y” 。唯一的两个字母的属性名称是InputEvent.up。否则，您不应该尝试在任何**GoJS**定义的对象上使用任何一个或两个字母的属性名称。

不要修改**GoJS**类的原型。  
只能使用[API中](http://gojs.net/latest/api/index.html)记录的属性和方法。

您也可以使用[TypeScript](http://www.typescriptlang.org/)来获得更好的“编译时”类型检查。**GoJS**的TypeScript定义文件名为“go.d.ts”，位于与“go.js”和“go-debug.js”库相同的目录中。在一些编辑器中，访问定义文件也大大提高了文档反馈，同时编辑了TypeScript代码。扩展类也已被翻译成TypeScript，可在../extensionsTS/。

您可以看到可以在[GoJS样品中](http://gojs.net/latest/samples/index.html)构建的各种图表。

在下一个介绍页面中，我们讨论[构建**GoJS**零件并将其添加到图表中。](http://gojs.net/latest/intro/buildingObjects.html)

**HTML互动**

本简介页面介绍了如何在Web应用程序中使用GoJS图表以及其他HTML元素。

对于通过GoJS工具操作调用和隐藏的自定义文本编辑器，上下文菜单和工具提示，最好使用HTMLInfo类。HTMLInfo在本页的第二部分描述。

在GoJS旁边使用HTML

使用HTML数据检查器编辑零件

一般来说，GoJS可以通过JavaScript来编程移动和修改GoJS对象和图表来与页面的其余部分进行交互。如果您尚未阅读有关与零件和模型进行编程交互的功能，则需要一个[GraphObject Manipulation教程](http://gojs.net/latest/learn/graphobject.html)。

为了帮助程序员开始使用HTML控件，我们实现了一个简单的[Data Inspector Extension](http://gojs.net/latest/extensions/DataInspector.html)，一个基于HTML的属性编辑器，显示并允许编辑所选零件的数据。

数据检查员主要通过"ChangedSelection" [图表监听器工作](http://gojs.net/latest/intro/events.html)。触发时，它填充HTML字段。编辑这些字段并单击，然后通过调用diagram.model.setDataProperty来更新所选的部件来更新模型。

jQuery和GoJS

GoJS不依赖于jQuery，但两者可以一起使用。该[标签示例](http://gojs.net/latest/samples/tabs.html)展示了如何使用jQuery的标签内GoJS。该[HTML互动样品](http://gojs.net/latest/samples/htmlInteraction.html)放置一个GoJS调色板一个jQuery移动窗口内，并修改内部另一当前选择的节点的数据检查。

jQuery通常设置$变量。如果要从我们的示例或文档中复制代码，请注意，我们通常会这样做： var $ = go.GraphObject.make;这样$我们的示例中的使用将构建GraphObject和其他GoJS对象。注意：在尝试构建**GoJS**对象时调用jQuery 会导致异常和隐含的错误。因此，您应该在本地分配$变量或使用不同的变量来构建**GoJS**对象。

HTML专注于图表

当浏览器元素集中时，它们将尽可能地滚动到视图中，这在某些Web应用程序中可能是不受欢迎的。您可以通过覆盖来禁用此效果Diagram.doFocus。

//保存并恢复窗口滚动值

//所以浏览器不会修改窗口滚动，当图集焦点

myDiagram.doFocus = function（）{

var x = window.scrollX || window.pageXOffset;

var y = window.scrollY || window.pageYOffset;

go.Diagram.prototype.doFocus.call（本）;

window.scrollTo（x，y）;

}

当图表对焦时，可以删除轮廓。这是一个CSS效果，而不是GoJS效果，可以通过从Diagram div中的所有canvas元素中删除CSS轮廓来删除：

/ \*影响myDiagramDiv内的所有画布元素\* /

#myDiagramDiv canvas {

大纲：无;

-webkit-tap-highlight-color：rgba（255,255,255,0）; / \* mobile webkit \* /

}

如果您的页面上不使用其他HTML Canvas元素，则可以对所有canvas元素执行此操作，以使其影响所有图，而不仅仅是特定DIV ID中的图。

HTMLInfo类

使用HTMLInfo类来显示自定义的HTML页面元素，例如由HTML组成的上下文菜单，工具提示或文本编辑器。

可以设置为包含的实例的属性HTMLInfo：

* TextEditingTool.defaultTextEditor
* TextBlock.textEditor
* GraphObject.contextMenu
* Diagram.contextMenu
* GraphObject.toolTip
* Diagram.toolTip

用法

当使用自定义功能替换GoJS功能时，主要关注的是何时显示和隐藏自定义内容。HTMLInfo使用程序员定义的两个可设置的函数并由GoJS调用：

* HTMLInfo.show，通过当应该激活的工具提示，ContextMenuTool，或TextEditingTool时显示的自定义的信息，例如GoJS调用。
* HTMLInfo.hide，当自定义信息完成后由GoJS调用，并且不再显示，例如在结束这些工具时。

代替设置HTMLInfo.hide，您可以将HTMLInfo.mainElement属性设置为您显示/隐藏的主HTML元素，HTMLInfo将通过调用以下方式自动隐藏提供的元素：

mainElement.style.display = "none";

HTMLInfo样本

* 文本编辑器：[自定义文本编辑器](http://gojs.net/latest/samples/customTextEditingTool.html)[对默认文本编辑](http://gojs.net/latest/extensions/TextEditor.html)[器进行抽样](http://gojs.net/latest/samples/customTextEditingTool.html)和[重新实现](http://gojs.net/latest/extensions/TextEditor.html)
* 上下文菜单：[自定义上下文菜单](http://gojs.net/latest/samples/customContextMenu.html)和[HTML灯箱上下文菜单](http://gojs.net/latest/samples/htmlLightBoxContextMenu.html)（重新实现默认触摸上下文菜单）
* 工具提示：[数据可视化工具提示](http://gojs.net/latest/samples/dataVisualization.html)

提示

对于工具提示，如果GraphObject.toolTip或Diagram.toolTip设置为实例HTMLInfo，GoJS呼吁HTMLInfo.show在ToolManager.showToolTip。工具提示延迟之后，GoJS将调用HTMLInfo.hide在ToolManager.hideToolTip。

下面是使用的例子HTMLInfo.show和HTMLInfo.hide，但HTMLInfo.hide非常简单，设置HTMLInfo.mainElement的提示DIV，而不是就足够了。

函数showToolTip（obj，图，工具）{

var toolTipDIV = document.getElementById（'toolTipDIV'）;

var pt = diagram.lastInput.viewPoint;

toolTipDIV.style.left =（pt.x + 10）+“px”;

toolTipDIV.style.top =（pt.y + 10）+“px”;

document.getElementById（'toolTipParagraph'）。textContent =“Tooltip for：”+ obj.data.key;

toolTipDIV.style.display =“block”;

}

函数hideToolTip（图，工具）{

var toolTipDIV = document.getElementById（'toolTipDIV'）;

toolTipDIV.style.display =“none”;

}

var myToolTip = $（go.HTMLInfo，{

显示：showToolTip，

隐藏：hideToolTip

/ \*

由于hideToolTip非常简单，

我们可以设置mainElement而不是设置hide：

mainElement：document.getElementById（'toolTipDIV'）

\* /

}）;

diagram.nodeTemplate =

$（go.Node，“Auto”，

{

toolTip：myToolTip

}，

$（go.Shape，“RoundedRectangle”，{strokeWidth：0}，

new go.Binding（“fill”，“color”）），

$（go.TextBlock，

{margin：8}，

new go.Binding（“text”，“key”））

）;

diagram.model = new go.GraphLinksModel（

[

{key：“Alpha”，color：“lightblue”}，

{key：“Beta”，color：“orange”}，

{key：“Gamma”，颜色：“lightgreen”}，

{key：“Delta”，color：“pink”}

]）;

<！ - 这必须添加为图的兄弟 - >

<div id =“toolTipDIV”style =“position：absolute; background：white; z-index：1000; display：none;”>

<p id =“toolTipParagraph”>工具提示

</ DIV>

上下文菜单

对于上下文菜单，ContextMenuTool.showContextMenu将调用HTMLInfo.show。ContextMenuTool.hideContextMenu将调用HTMLInfo.hide。

// Assign an HTMLInfo to the Diagram:

myDiagram.contextMenu = $(go.HTMLInfo, {

show: showContextMenu,

hide: hideContextMenu

});

function showContextMenu(obj, diagram, tool) {

// Show the context menu HTML element:

SomeDOMElement.style.display = "block";

// Also show relevant buttons given the current state

// and the GraphObject obj; if null, the context menu is for the whole Diagram

}

function hideContextMenu() {

SomeDOMElement.style.display = "none";

}

function buttonClick() {

// do some action when a context menu button is clicked

// then:

myDiagram.currentTool.stopTool();

}

文本编辑器

对于自定义文本编辑器，TextEditingTool.doActivate将调用HTMLInfo.show。TextEditingTool.doDeactivate将调用HTMLInfo.hide。

用作文本编辑器的HTMLInfos还必须定义一个HTMLInfo.valueFunction。当TextEditingTool.acceptText被调用时，GoJS将调用HTMLInfo.valueFunction并使用返回值作为TextEditingTool完成的值。

下面的示例构建了一个HTMLInfo，它使用HTMLInfo.show并HTMLInfo.hide动态地从页面添加，填充和删除HTML元素。

//图设置。HTMLInfo设置在此代码块的末尾。

diagram.nodeTemplate =

$（go.Node，“Auto”，

$（go.Shape，“RoundedRectangle”，{strokeWidth：0}，

new go.Binding（“fill”，“color”）），

$（go.TextBlock，

{editable：true，margin：8，choices：['Alpha'，'Beta'，'Gamma'，'Delta']}，

new go.Binding（“text”））

）;

diagram.model = new go.GraphLinksModel（

[

{text：“Alpha”，color：“lightblue”}，

{text：“Beta”，color：“orange”}，

{text：“Gamma”，color：“lightgreen”}，

{text：“Delta”，color：“pink”}

]）;

//创建一个HTMLInfo并动态创建一些HTML来显示/隐藏

var customEditor = new go.HTMLInfo（）;

var customSelectBox = document.createElement（“select”）;

customEditor.show = function（textBlock，diagram，tool）{

if（！（textBlock instanceof go.TextBlock））return;

//填充选择框：

customSelectBox.innerHTML =“”;

//此示例假定textBlock.choices不为空

var list = textBlock.choices;

for（var i = 0; i <list.length; i ++）{

var op = document.createElement（“option”）;

op.text = list [i];

op.value = list [i];

customSelectBox.add（op，null）;

}

//填写列表后，设置值：

customSelectBox.value = textBlock.text;

//当用户按下一个键时，做一些不同的事情

customSelectBox.addEventListener（“keydown”，function（e）{

var keynum = e.which;

if（keynum == 13）{//接受输入

tool.acceptText（go.TextEditingTool.Enter）;

返回;

} else if（keynum == 9）{//接受Tab

tool.acceptText（go.TextEditingTool.Tab）;

e.preventDefault（）;

返回假

} else if（keynum === 27）{//取消在Esc

tool.doCancel（）;

if（tool.diagram）tool.diagram.focus（）;

}

}，false）;

var loc = textBlock.getDocumentPoint（go.Spot.TopLeft）;

var pos = diagram.transformDocToView（loc）;

customSelectBox.style.left = pos.x +“px”;

customSelectBox.style.top = pos.y +“px”;

customSelectBox.style.position ='absolute';

customSelectBox.style.zIndex = 100; //将其放在图的前面

diagram.div.appendChild（customSelectBox）;

}

customEditor.hide = function（图，工具）{

diagram.div.removeChild（customSelectBox）;

}

//这对于用作文本编辑器的HTMLInfo实例是必需的

customEditor.valueFunction = function（）{return customSelectBox.value; }

//设置HTMLInfo：

diagram.toolManager.textEditingTool.defaultTextEditor = customEditor;

**活动**

**GoJS** 有三种基本类型的事件： [DiagramEvent](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent%3C/font%3E%3C/font%3E.html)，[InputEvent](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EInputEvent%3C/font%3E%3C/font%3E.html)和[ChangedEvent](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EChangedEvent%3C/font%3E%3C/font%3E.html)。本页讨论前两个; 请参阅最近一次[活动](http://gojs.net/latest/intro/changedEvents.html)的[更改](http://gojs.net/latest/intro/changedEvents.html)事件。

图事件

[DiagramEvent](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent%3C/font%3E%3C/font%3E.html)表示一般用户启动的图的更改。您可以通过调用[Diagram.addDiagramListener](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#addDiagramListener</font></font>)来注册图事件处理程序。每种图事件都以其名称进行区分。

当前定义的图事件名称包括：

* “ **AnimationStarting** ”，动画即将开始。
* “ **AnimationFinished** ”，刚刚完成的动画。
* “ **BackgroundSingleClicked** ”，当鼠标左键单击鼠标在图的后台单击时，不在零件上
* “ **BackgroundDoubleClicked** ”，当鼠标左键双击发生在图的后台，而不是零件
* “ **BackgroundContextClicked** ”，当鼠标右键单击发生在图的后台时，不在零件上
* “ **更改选择** ”，在[Diagram.selection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#selection</font></font>)集合更改之前
* “ **ChangedSelection** ”，后[Diagram.selection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#selection</font></font>)收集已经改变
* “ **ClipboardChanged** ”，当[CommandHandler.copySelection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ECommandHandler.html#copySelection</font></font>)已更改剪贴板（但图未修改）
* “ **ClipboardPasted** ”，当[CommandHandler.pasteSelection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ECommandHandler.html#pasteSelection</font></font>)将零件从剪贴板复制到图中时; 新零件保存在[Diagram.selection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#selection</font></font>)集合中
* “ **DocumentBoundsChanged** ”，当[Diagram.documentBounds](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#documentBounds</font></font>)更改值时
* “ **ExternalObjectsDropped** ”，当[DraggingTool](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDraggingTool%3C/font%3E%3C/font%3E.html)将零件从另一个图表复制到图中时; 新零件保存在[Diagram.selection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#selection</font></font>)集合中
* “ **InitialLayoutCompleted** ”，当第一个图表布局在图表发生重大变化后完成
* “ **LayoutCompleted** ”，每当图表布局完成
* “ **LinkDrawn** ”，当[LinkingTool](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ELinkingTool%3C/font%3E%3C/font%3E.html)已经完成创建一个新的链接，这是[DiagramEvent.subject](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent.html#subject</font></font>)
* “ **LinkRelinked** ”，当重新链接[工具](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3E%E5%B7%A5%E5%85%B7%3C/font%3E%3C/font%3E.html)已经完成重新连接现有的链接，这是[DiagramEvent.subject](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent.html#subject</font></font>)
* “ **LinkReshaped** ”，当[LinkReshapingTool](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ELinkReshapingTool%3C/font%3E%3C/font%3E.html)已经完成重新路由一个链接，这是[DiagramEvent.subject](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent.html#subject</font></font>)
* “ **修改** ”，[Diagram.isModified](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#isModified</font></font>)属性已设置为新值; 不要在事件侦听器中修改图或其模型
* “ **PartCreated** ”，当[ClickCreatingTool](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EClickCreatingTool%3C/font%3E%3C/font%3E.html)插入了一个新的Part，它是[DiagramEvent.subject](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent.html#subject</font></font>)
* “ **PartResized** ”，当[ResizingTool](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EResizingTool%3C/font%3E%3C/font%3E.html)已经调整了GraphObject，它是[DiagramEvent.subject](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent.html#subject</font></font>)
* “ **PartRotated** ”，当[RotatingTool](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ERotatingTool%3C/font%3E%3C/font%3E.html)已经旋转一个GraphObject，这是[DiagramEvent.subject](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent.html#subject</font></font>)
* “ **ObjectSingleClicked** ”，当一个鼠标左键单击发生在GraphObject上，这是[DiagramEvent.subject](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent.html#subject</font></font>)
* “ **ObjectDoubleClicked** ”，当鼠标左键双击发生在GraphObject上时，它是[DiagramEvent.subject](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent.html#subject</font></font>)
* “ **ObjectContextClicked** ”，当鼠标右键单击发生在GraphObject上时，它是[DiagramEvent.subject](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent.html#subject</font></font>)
* “ **SelectionMoved** ”，当[DraggingTool](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDraggingTool%3C/font%3E%3C/font%3E.html)移动了一些零件; 移动的零件保存在[Diagram.selection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#selection</font></font>)集合中
* “ **SelectionCopied** ”，当[DraggingTool](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDraggingTool%3C/font%3E%3C/font%3E.html)复制了一些零件; 复制的零件保存在[Diagram.selection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#selection</font></font>)集合中
* “ **SelectionDeleting** ”，前[CommandHandler.deleteSelection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ECommandHandler.html#deleteSelection</font></font>)删除[Diagram.selection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#selection</font></font>)配件
* “ **SelectionDeleted** ”，之后[CommandHandler.deleteSelection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ECommandHandler.html#deleteSelection</font></font>)已经删除了[Diagram.selection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#selection</font></font>)，它现在是空的
* “ **SelectionGrouped** ”，后[CommandHandler.groupSelection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ECommandHandler.html#groupSelection</font></font>)具有创建从一个新的组[Diagram.selection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#selection</font></font>)，其现在持有的新集团
* “ **SelectionUngrouped** ”，在[CommandHandler.ungroupSelection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ECommandHandler.html#ungroupSelection</font></font>)已删除所选组并将成员零件保留为新的[Diagram.selection](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#selection</font></font>)
* “ **SubGraphCollapsed** ”，当[CommandHandler.collapseSubGraph](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ECommandHandler.html#collapseSubGraph</font></font>)已折叠选定的组时，使其成员部件不可见
* “ **SubGraphExpanded** ”，当[CommandHandler.expandSubGraph](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ECommandHandler.html#expandSubGraph</font></font>)扩展了选定的组，使其成员部件可见
* “ **TextEdited** ”，当[TextEditingTool](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ETextEditingTool%3C/font%3E%3C/font%3E.html)完成就地文本编辑; 修改的TextBlock是[DiagramEvent.subject](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent.html#subject</font></font>)
* “ **TreeCollapsed** ”，当[CommandHandler.collapseTree](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ECommandHandler.html#collapseTree</font></font>)已经使用子树折叠选定的节点时，使子树节点和链接不可见
* “ **TreeExpanded** ”，当[CommandHandler.expandTree](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ECommandHandler.html#expandTree</font></font>)使用子树扩展选定节点时，使子树节点和链接可见
* “ **ViewportBoundsChanged** ”，当[Diagram.viewportBounds](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#viewportBounds</font></font>)改变了值

有关更多详细信息，请阅读[DiagramEvent](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent%3C/font%3E%3C/font%3E.html)的文档。

图表事件不一定对应于鼠标事件或键盘事件或触摸事件。也不必对应于更改图表的模型-跟踪这种变化，使用[Model.addChangedListener](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EModel.html#addChangedListener</font></font>)或[Diagram.addModelChangedListener](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#addModelChangedListener</font></font>)。图表事件只会发生，因为用户可能间接地做了一些事情。

除了DiagramEvent侦听器之外，还有一些情况是检测到这种变化是足够常见的，以保证具有事件处理程序的属性。因为这些事件不一定对应于任何特定的输入或图表事件，这些事件处理程序具有特定于这种情况的自定义参数。

一个非常常见的事件属性是[GraphObject.click](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EGraphObject.html#click</font></font>)，如果非空是一个在用户点击该对象时调用的函数。这通常用于指定“Button”的行为，但它和另一个“click”事件属性“doubleClick”和“contextClick”可用于任何GraphObject。

另一个常见的事件属性是[Part.selectionChanged](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EPart.html#selectionChanged</font></font>)，如果[Part.isSelected](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EPart.html#isSelected</font></font>)更改，则调用（如果非空）。在这种情况下，事件处理函数传递一个参数，即Part。不需要额外的参数，因为该函数可以检查[Part.isSelected](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EPart.html#isSelected</font></font>)的当前值来决定要做什么。

[ChangedEvent](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EChangedEvent%3C/font%3E%3C/font%3E.html) 模型比依赖于[DiagramEvent](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagramEvent%3C/font%3E%3C/font%3E.html)更完整可靠。例如，当代码添加到图表的链接时，不会引发“LinkDrawn”DiagramEvent。仅当用户使用[LinkingTool](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ELinkingTool%3C/font%3E%3C/font%3E.html)绘制新链接时，才引发DiagramEvent 。此外，链接尚未路由，因此[Link.points](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3ELink.html#points</font></font>)不会被计算。事实上，创建一个新的链接可能使[布局](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3E%E5%B8%83%E5%B1%80%3C/font%3E%3C/font%3E.html)无效，所以所有的节点可能会在不久的将来被移动。

有时您希望在用户更改图表时更新数据库。通常你会想实现一个[型号](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3E%E5%9E%8B%E5%8F%B7%3C/font%3E%3C/font%3E.html) [ChangedEvent](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EChangedEvent%3C/font%3E%3C/font%3E.html)监听器，通过调用[Model.addChangedListener](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EModel.html#addChangedListener</font></font>)或[Diagram.addModelChangedListener](http://gojs.net/latest/api/symbols/%3Cfont%3E%3Cfont%3EDiagram.html#addModelChangedListener</font></font>)，即注意到了更改模型，并决定如何在数据库中记录。请参阅[更改事件](http://gojs.net/latest/intro/changedEvents.html)和[更新演示](http://gojs.net/latest/samples/UpdateDemo.html)的讨论。

此示例演示如何处理几个图事件：**“ObjectSingleClicked”**， **“BackgroundDoubleClicked”**和**“ClipboardPasted”**。

function showMessage （s） {

document.getElementById（ “diagramEventsMsg”）.textContent = s;

}

show.addDiagramListener（“ObjectSingleClicked”，

function （e） {

var part = e.subject.part;

if（！（part instanceof go.Link））showMessage（“Clicked” + part.data.key）;

}）;

diagram.addDiagramListener（“BackgroundDoubleClicked”，

function （e） { showMessage（“double- click at at” + e.diagram.lastInput.documentPoint）;}）;

diagram.addDiagramListener（“ClipboardPasted”，

function （e） { showMessage（“Pasted” + e.diagram.selection.count + “parts”）;}）;

var nodeDataArray = [

{key：“Alpha” }，

{key：“Beta”，组：“Omega” }，

{key：“Gamma”，组：“Omega” }，

{key：“Omega”，isGroup：true }，

{key：“Delta” }

]。

var linkDataArray = [

{from：“Alpha”，to：“Beta” }， //从组外面的内部

{从：“Beta”到：“Gamma” } //这个链接是组的成员

{from：“Omega”，to：“Delta” } //从组到节点

]。diagram.model

= new go.GraphLinksModel（nodeDataArray，linkDataArray）;

（信息）

输入事件

当发生低级HTML DOM事件时，**GoJS**将键盘/鼠标/触摸事件信息规范化为一个新的[InputEvent](http://gojs.net/latest/api/symbols/InputEvent.html)，可以传递给各种事件处理方法并保存供以后检查。

一个InputEvent的保持[InputEvent.key](http://gojs.net/latest/api/symbols/InputEvent.html#key)键盘事件，该[InputEvent.button](http://gojs.net/latest/api/symbols/InputEvent.html#button)鼠标事件，该[InputEvent.viewPoint](http://gojs.net/latest/api/symbols/InputEvent.html#viewPoint)鼠标和触摸事件，并[InputEvent.modifiers](http://gojs.net/latest/api/symbols/InputEvent.html#modifiers)键盘和鼠标事件。

该图的事件处理程序也记录[InputEvent.documentPoint](http://gojs.net/latest/api/symbols/InputEvent.html#documentPoint)，这是[InputEvent.viewPoint](http://gojs.net/latest/api/symbols/InputEvent.html#viewPoint)文件在鼠标事件的时间坐标，以及[InputEvent.timestamp](http://gojs.net/latest/api/symbols/InputEvent.html#timestamp)，记录了事件发生的时间（毫秒）。

InputEvent类还为特定类型的事件提供了许多方便的属性。示例包括[InputEvent.control](http://gojs.net/latest/api/symbols/InputEvent.html#control)（如果控制键已被按下）和[InputEvent.left](http://gojs.net/latest/api/symbols/InputEvent.html#left)（如果按下 了左/主鼠标按钮）。

一些工具在鼠标点找到“当前” [GraphObject](http://gojs.net/latest/api/symbols/GraphObject.html)。这被记住为[InputEvent.targetObject](http://gojs.net/latest/api/symbols/InputEvent.html#targetObject)。

更高级别的输入事件

一些工具检测一系列输入事件，以构成一些更抽象的用户事件。示例包括“点击”（鼠标向上和向下非常接近）和“悬停”（静止鼠标一段时间）。这些工具将在鼠标点调用当前[GraphObject](http://gojs.net/latest/api/symbols/GraphObject.html)的事件处理程序（如果有的话）。事件处理程序作为对象上属性的值保存。然后它还将事件“Bubble”到[GraphObject.panel](http://gojs.net/latest/api/symbols/GraphObject.html#panel)的链，直到它以[Part](http://gojs.net/latest/api/symbols/Part.html)结尾。这允许在[Panel](http://gojs.net/latest/api/symbols/Panel.html)上声明“点击”事件处理程序，并且即使点击实际发生在面板内部的元素上也适用。如果鼠标点没有对象，则图表上会出现该事件。

点击式事件属性包括[GraphObject.click](http://gojs.net/latest/api/symbols/GraphObject.html#click)，[GraphObject.doubleClick](http://gojs.net/latest/api/symbols/GraphObject.html#doubleClick)和[GraphObject.contextClick](http://gojs.net/latest/api/symbols/GraphObject.html#contextClick)。它们也出现在没有GraphObject的情况下 - 事件在图的背景中发生： [Diagram.click](http://gojs.net/latest/api/symbols/Diagram.html#click)，[Diagram.doubleClick](http://gojs.net/latest/api/symbols/Diagram.html#doubleClick)和[Diagram.contextClick](http://gojs.net/latest/api/symbols/Diagram.html#contextClick)。这些都是可以设置为事件处理程序的函数的属性。这些事件都是由鼠标事件和触摸事件引起的。

鼠标类似的事件属性包括[GraphObject.mouseEnter](http://gojs.net/latest/api/symbols/GraphObject.html#mouseEnter)，[GraphObject.mouseOver](http://gojs.net/latest/api/symbols/GraphObject.html#mouseOver)和[GraphObject.mouseLeave](http://gojs.net/latest/api/symbols/GraphObject.html#mouseLeave)。但只有[Diagram.mouseOver](http://gojs.net/latest/api/symbols/Diagram.html#mouseOver)适用于图表。

类似于Hover的事件属性包括[GraphObject.mouseHover](http://gojs.net/latest/api/symbols/GraphObject.html#mouseHover)和[GraphObject.mouseHold](http://gojs.net/latest/api/symbols/GraphObject.html#mouseHold)。等效图表属性是[Diagram.mouseHover](http://gojs.net/latest/api/symbols/Diagram.html#mouseHover)和[Diagram.mouseHold](http://gojs.net/latest/api/symbols/Diagram.html#mouseHold)。

还有用于拖动操作的事件属性：[GraphObject.mouseDragEnter](http://gojs.net/latest/api/symbols/GraphObject.html#mouseDragEnter)，[GraphObject.mouseDragLeave](http://gojs.net/latest/api/symbols/GraphObject.html#mouseDragLeave)和[GraphObject.mouseDrop](http://gojs.net/latest/api/symbols/GraphObject.html#mouseDrop)。这些适用于固定物体，而不是物体被拖动。当触摸事件拖动时，也不会只是鼠标事件发生。

此示例演示如何处理三个较高级别的输入事件：单击节点和进入/离开组。

function showMessage （s） {

document.getElementById（ “inputEventsMsg”）.textContent = s;

}

diagram.nodeTemplate =

$（go.Node，“Auto”，

$（go.Shape，“Ellipse”，{fill：“white” }），

$（go.TextBlock，

new go.Binding（“text”，“key”）），

{click：function （e，obj） { showMessage（“ clicked ” + obj.part.data.key）; }}

）;

diagram.groupTemplate =

$（go.Group，“Vertical”，

$（go.TextBlock，

{alignment：go.Spot.Left，font：“Bold 12pt Sans-Serif” }，

new go.Binding（“text”，“key”）），

$（go.Panel，“Auto”，

$（go.Shape，“ RoundedRectangle ”，

{name：“SHAPE”，

参数1：14，

填写：“rgba（128,128,128,0.33）” }），

$（go.Placeholder，{padding：5 }）

）

{mouseEnter：function （e，obj，prev） { //更改组的背景画刷

var shape = obj.part.findObject（“SHAPE”）;

if（shape）shape.fill = “red” ;

}，

mouseLeave：function （e，obj，next） { //恢复到原始的画笔

var shape = obj.part.findObject（“SHAPE”）;

if（shape）shape.fill = “rgba（128,128,128,0.33）” ;

}}）;

var nodeDataArray = [

{key：“Alpha” }，

{key：“Beta”，组：“Omega” }，

{key：“Gamma”，组：“Omega” }，

{key：“Omega”，isGroup：true }，

{key：“Delta” }

]。

var linkDataArray = [

{from：“Alpha”，to：“Beta” }， //从组外面的内部

{从：“Beta”到：“Gamma” } //这个链接是组的成员

{from：“Omega”，to：“Delta” } //从组到节点

]。diagram.model

= new go.GraphLinksModel（nodeDataArray，linkDataArray）;

（信息）

点击并选择

此示例演示了“点击”和“selectionChanged”事件：

function showMessage （s） {

document.getElementById（ “changeMethodsMsg”）.textContent = s;

}

diagram.nodeTemplate =

$（go.Node，“Auto”，

{selectionAdorned：false }，

$（go.Shape，“Ellipse”，{fill：“white” }），

$（go.TextBlock，

new go.Binding（“text”，“key”）），

{

点击：function （e，obj） { showMessage（“ clicked ” + obj.part.data.key）; }，

selectionChanged：function （part） {

var shape = part.elt（0）;

shape.fill = part.isSelected？“红”：“白” ;

}

}

）;

var nodeDataArray = [

{key：“Alpha” }，{key：“Beta” }，{key：“Gamma” }

]。

var linkDataArray = [

{from：“Alpha”，to：“Beta” }，

{from：“Beta”，to：“Gamma” }

]。diagram.model

= new go.GraphLinksModel（nodeDataArray，linkDataArray）;

点击测试版

尝试Ctrl-A选择所有内容。注意[GraphObject.click](http://gojs.net/latest/api/symbols/GraphObject.html#click)事件属性和[Part.selectionChanged](http://gojs.net/latest/api/symbols/Part.html#selectionChanged)事件属性之间的区别。两者都是当节点发生了某些事情时被调用的方法。所述[GraphObject.click](http://gojs.net/latest/api/symbols/GraphObject.html#click)当用户点击的节点，这恰好选择的节点上发生。但是即使没有点击事件甚至任何鼠标事件 - 即由于对该节点的属性更改，[Part.selectionChanged](http://gojs.net/latest/api/symbols/Part.html#selectionChanged)也会发生。

# 工具提示

**GoJS**提供了一种为任何对象或图表背景创建自定义工具提示的方法。

工具提示是当鼠标悬停在其GraphObject.toolTip设置的对象上时显示的“ 装饰”。工具提示部分被绑定到与部件本身相同的数据。

请参阅[样品索引中](http://gojs.net/latest/samples/index.html#tooltips)使用工具提示的[示例](http://gojs.net/latest/samples/index.html#tooltips)。

在此示例中，每个节点的GraphObject.toolTip属性设置为通过正常数据绑定显示data.color属性的部分。该图通过设置Diagram.toolTip获取自己的工具提示。

diagram.nodeTemplate =

$（go.Node，“Auto”，

$（go.Shape，“RoundedRectangle”，

{fill：“white”}，

new go.Binding（“fill”，“color”）），

$（go.TextBlock，{margin：5}，

new go.Binding（“text”，“key”）），

{

toolTip：//为每个以文本形式显示颜色的节点定义工具提示

$（go.Adornment，“Auto”，

$（go.Shape，{fill：“#FFFFCC”}），

$（go.TextBlock，{margin：4}，

new go.Binding（“text”，“color”））

）//装饰结束

}

）;

//生成图表工具提示的内容的函数

function diagramInfo（model）{

返回“Model：\ n”+ model.nodeDataArray.length +“nodes”，+

model.linkDataArray.length +“links”;

}

//为图的背景提供一个工具提示，当不在任何部分时

diagram.toolTip =

$（go.Adornment，“Auto”，

$（go.Shape，{fill：“#CCFFCC”}），

$（go.TextBlock，{margin：4}，

//使用转换器显示关于图模型的信息

new go.Binding（“text”，“”，diagramInfo））

）;

var nodeDataArray = [

{key：“Alpha”，color：“lightblue”}，

{key：“Beta”，color：“pink”}

]。

var linkDataArray = [

{from：“Alpha”，to：“Beta”}

]。

diagram.model = new go.GraphLinksModel（nodeDataArray，linkDataArray）;

尝试将鼠标暂停在每个节点上或在图的后台。如果您复制一些零件，您将看到图表的工具提示显示有关图表的更新信息。

您可以通过设置ToolManager.hoverDelay来更改鼠标在工具提示出现之前等待静止多长时间。例如，当初始化图时，"toolManager.hoverDelay": 600将延迟更改为1/6秒的1/6秒。

您可以通过设置ToolManager.toolTipDuration来更改工具提示保持可见的时间。例如，"toolManager.toolTipDuration": 10000将可见时间更改为10秒。

### 定位

有两种方法来自定义工具提示相对于装饰的GraphObject的定位。一种方法是覆盖ToolManager.positionToolTip。另一种方法是有提示装饰品包括一个占位符。占位符的位置与装饰对象具有相同的大小和位置。

//这是一个普通的Node模板，它也为它定义了一个toolTip

diagram.nodeTemplate =

$（go.Node，“Auto”，

$（go.Shape，“RoundedRectangle”，

{fill：“white”}，

new go.Binding（“fill”，“color”）），

$（go.TextBlock，{margin：5}，

new go.Binding（“text”，“key”）），

{

toolTip：//为每个节点定义一个工具提示

$（go.Adornment，“Spot”，//它周围有几个标签

{background：“transparent”}，//避免在鼠标移动时隐藏工具提示

$（go.Placeholder，{padding：5}），

$（go.TextBlock，

{alignment：go.Spot.Top，alignmentFocus：go.Spot.Bottom，stroke：“red”}，

new go.Binding（“text”，“key”，function（s）{return“key：”+ s;}）），

$（go.TextBlock，“Bottom”，

{alignment：go.Spot.Bottom，alignmentFocus：go.Spot.Top，stroke：“red”}，

new go.Binding（“text”，“color”，function（s）{return“color：”+ s;}））

）//结束装饰

}

）;

diagram.initialContentAlignment = go.Spot.Center;

var nodeDataArray = [

{key：“Alpha”，color：“lightyellow”}，

{key：“Beta”，color：“orange”}

]。

var linkDataArray = [

{from：“Alpha”，to：“Beta”}

]。

diagram.model = new go.GraphLinksModel（nodeDataArray，linkDataArray）;

注意如何装饰品实现工具提示使用“透明的”背景，这样的提示不会自动鼠标移动时去除。

## HTML工具提示

可以使用HTML而不是使用HTMLInfo类的Adornment来定义自定义工具提示。该[数据可视化样品](http://gojs.net/latest/samples/dataVisualization.html)示出了这样的工具提示。请参阅[HTML交互](http://gojs.net/latest/intro/HTMLInteraction.html)以进行更多讨论。