【mxGraph】源码学习：（5）mxGraph

> 由于mxGraph源文件有一万多行，且涉及很多其它源文件，所以重点在于了解mxGraph的作用、结构以及定义了哪些方法

1. 概览

1.1 作用

mxGraph继承自mxEventSource以实现基于Web的图形组件的功能性方面。要激活平移和连接，使用setPanning和setConnectable，对于框线选择，必须创建一个新的mxRubberband实例。默认情况下，以下监听器添加到mouseListeners：

1. tooltipHandler：显示工具提示的mxTooltipHandler
2. panningHandler：用于平移和弹出菜单的mxPanningHandler
3. connectionHandler：用于创建连接的mxConnectionHandler
4. graphHandler：用于移动和克隆cell的mxGraphHandler

如果启用了这些监听器，则将按上述顺序调用它们。

1.2 背景图片

要显示背景图像，使用setBackgroundImage设置图像URL和宽高。更改上述值之一后，应调用视图的mxGraphView.validate。

1.3 cell图像

要在cell中使用图像，必须在默认vertex样式（或任何命名样式）中指定形状。可能的形状是mxConstants.SHAPE_IMAGE和mxConstants.SHAPE_LABEL。更改默认vertex样式中使用的形状的代码，如下所示：

```
var style = graph.getStylesheet().getDefaultVertexStyle();
style[mxConstants.STYLE_SHAPE] = mxConstants.SHAPE_IMAGE;
```

对于默认vertex样式，可以使用mxConstants.STYLE_IMAGE键和图像URL作为值在cell样式中指定要显示的图像，例如：

```
image=http://www.example.com/image.gif
```

对于命名样式，stylename必须是cell样式的第一个元素：

```
stylename;image=http://www.example.com/image.gif
```

cell样式可以添加任意数量的键值对，用分号隔开，如下所示：

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

1.4 标签

cell标签由getLabel定义，如果labelsVisible为true ，则使用convertValueToString。如果必须将标签呈现为HTML标记，则isHtmlLabel应对相应的cell格返回true。如果所有标签都包含HTML标记，则htmlLabels可以设置为true。注意：启用HTML标签可能存在安全风险（请

参阅手册中的安全性部分）。

如果标签需要包装，那么isHtmlLabel和isWrapping必须为其标签应该被包装的cell格返回true。请参阅isWrapping示例。

如果需要剪切以将HTML标签的显示保持在其vertex的边界内，则isClipping应对相应的cell格返回true。

默认情况下，edge标签是可移动的，vertex标签是固定的。这可以通过设置edgeLabelsMovable和vertexLabelsMovable，或通过覆盖isLabelMovable来更改。

1.5 就地编辑

通过双击或键入F2启动就地编辑。以编程方式，edit用于检查cell是否可编辑（isCellEditable）并调用startEditingAtCell，它调用mxCellEditor.startEditing。编辑器使用getEditingValue返回的值作为编辑值。

就地编辑后，labelChanged被调用，调用mxGraphModel.setValue，进而调用mxGraphModel.valueForCellChanged通过mxValueChange。

触发就地编辑的事件将传递给cellEditor，后者可能会根据事件类型或鼠标位置采取特殊操作，也会传递给getEditingValue。然后将事件传递回事件处理函数，该函数可以基于触发事件执行特定动作。

1.6 提示

工具提示由getTooltip实现，如果cell位于鼠标指针下，则调用getTooltipForCell。默认实现检查cell是否具有getTooltip函数，如果存在则调用它。因此，为了提供自定义工具提示，cell必须提供getTooltip函数，或者必须覆盖上述两个函数之一。

通常对于自定义cell工具提示，后一个函数被覆盖如下：

```
graph.getTooltipForCell = function(cell) {
  var label = this.convertValueToString(cell);
  return 'Tooltip for '+label;
}
```

使用配置文件时，使用以下项在mxGraph部分中覆盖该函数：

```
<add as="getTooltipForCell"><![CDATA[
  function(cell) {
    var label = this.convertValueToString(cell);
    return 'Tooltip for '+label;
  }
]]></add>
```

this是指实现中的graph，例如为了检查cell是否是edge，使用this.getModel().isEdge(cell)。

要替换getTooltipForCell的默认实现（而不是替换特定实例上的函数），在加载js文件之后，但在使用mxGraph创建新的mxGraph实例之前，使用以下代码：

```
mxGraph.prototype.getTooltipForCell = function(cell) {
  var label = this.convertValueToString(cell);
  return 'Tooltip for '+label;
}
```

1.7 形状和样式

在示例中演示了新形状的实现。假设已经实现了一个名为BoxShape的自定义形状，想要用它来绘制vertex。要使用此形状，必须首先在cell渲染器中注册，如下所示：

```
mxCellRenderer.registerShape('box', BoxShape);
```

该代码在graph的cell渲染器中的名称框下注册BoxShape构造函数。现在可以使用样式定义中的shape-key来引用形状。（cell渲染器包含一组其他形状，即每个常量一个，在mxConstants中具有SHAPE前缀）

样式是键值对的集合，样式表是命名样式的集合。名称由cellstyle引用，它以mxCell.style存储，格式如 `[stylename;|key=value;]` 。该字符串被解析为键值对的集合，其中键被字符串中的值覆盖。

引入新形状时，必须在样式表中使用注册形状的名称。有三种方法可以做到这一点：

1. 通过更改默认样式，使所有vertex都使用新形状
2. 通过定义新样式，只有具有相应cell样式的vertex才会使用新形状
3. 通过在cellstyle的可选键值对列表中使用shape=box覆盖

在第一种情况下，获取和修改vertex的默认样式的代码如下：

```
var style = graph.getStylesheet().getDefaultVertexStyle();
style[mxConstants.STYLE_SHAPE] = 'box';
```

该代码采用默认的vertex样式，该样式用于没有特定cell样式的所有vertex，并在原位修改shape-key的值以使用新的BoxShape绘制vertex。这是通过在第二行中指定框值来完成的，该值是指cell渲染器中BoxShape的名称。

在第二种情况下，创建一组键值对，然后以新名称添加到样式表中。为了区分shapename和stylename，我们将使用boxstyle作为stylename：

```
var style = new Object();
style[mxConstants.STYLE_SHAPE] = 'box';
style[mxConstants.STYLE_STROKECOLOR] = '#000000';
style[mxConstants.STYLE_FONTCOLOR] = '#000000';
graph.getStylesheet().putCellStyle('boxstyle', style);
```

该代码将一个名为boxstyle的新样式添加到样式表中。要将此样式与cell一起使用，必须从cellstyle引用它，如下所示：

```
var vertex = graph.insertVertex(parent, null, 'Hello, World!', 20, 20, 80, 20,
             'boxstyle');
```

总而言之，必须在mxCellRenderer中使用唯一名称注册每个新形状。然后，该名称将用作默认或自定义样式中shape-key的值。如果有多个自定义形状，则每个形状应该有一个单独的样式。

1.8 继承样式

对于fill-，stroke-，gradient-和indicatorColors，可以使用特殊关键字。其中一种颜色的inherit关键字将继承父cell中相同键的颜色。swimlane关键字执行相同操作，但继承自祖先层次结构中最近的swimlane。最后，指示的关键字将使用指标的颜色作为给定键的颜色。

1.9 滚动条

containers overflow CSS属性定义滚动条是否用于显示graph。对于'auto'或'scroll'的值，将显示滚动条。请注意，resizeContainer标志通常不与滚动条一起使用，因为它会在每次更改后调整容器大小以匹配graph的大小。

1.10 多重性和验证

要控制mxGraph中可能的连接，请使用getEdgeValidationError。函数的默认实现使用多重性，即mxMultiplicity数组。使用此类可以建立简单的多重性，由graph强制执行。

mxMultiplicity使用mxCell.is来确定它适用的终端。mxCell.is的默认实现与DOM节点（XML节点）一起使用，并检查给定的类型参数是否与节点的nodeName匹配（不区分大小写）。可选地，可以指定也检查的属性名和值。

只要edge的连接发生更改，就会调用getEdgeValidationError。如果edge无效，则返回空字符串或错误消息，如果edge有效，则返回null。如果返回的字符串不为空，则显示为错误消息。

mxMultiplicity允许指定终端与其可能的邻居之间的多重性。例如，如果任何矩形只能连接到最多两个圆圈，则可以将以下规则添加到多重性：

```
graph.multiplicities.push(new mxMultiplicity(
  true, 'rectangle', null, null, 0, 2, ['circle'],
  'Only 2 targets allowed',
  'Only shape targets allowed'));
```

每当矩形连接到两个以上的圆圈时，这将显示第一条错误消息，如果矩形连接到除圆形之外的任何东西，则显示第二条错误消息。

对于某些多重性，例如最少1个连接，在创建cell时不能强制执行（除非cell与连接一起创建），mxGraph提供validate，它检查所有cell的所有多重性并显示相应的错误消息在cell上的叠加图标中。

如果cell已折叠且包含验证错误，则会在折叠的cell上附加相应的警告图标。

1.11 自动布局

对于自动布局，在mxLayoutManager中提供了getLayout钩子。可以重写它以返回给定cell的子节点的布局算法。

1.12 未连接的edge

所有开关的默认值都旨在满足一般图表绘图应用程序的要求。一组非常典型的设置可避免未连接的edge如下：

```
graph.setAllowDanglingEdges(false);
graph.setDisconnectOnMove(false);
```

将cloneInvalidEdges开关设置为true是可选的。此开关控制是否在复制，粘贴或克隆拖动后插入edge（如果它们无效）。例如，如果在没有选择相应的终端且allowDanglingEdges为false的情况下复制或控制拖动edge，则edge无效，在这种情况下，如果开关为假，则不会克隆edge。

1.13 输出

要为graph生成XML表示，可以使用以下代码：

```
var enc = new mxCodec(mxUtils.createXmlDocument());
var node = enc.encode(graph.getModel());
```

这将生成一个XML节点，而不是使用DOM API处理或使用以下代码转换为字符串表示：

```
var xml = mxUtils.getXml(node);
```

要获取格式化字符串，可以使用mxUtils.getPrettyXml。

此字符串现在可以存储在本地持久存储中（例如使用Google Gears），也可以使用mxUtils.post将其传递到后端，如下所示。url变量是Java servlet、PHP页面或HTTP处理程序的URL，具体取决于服务器。

```
var xmlString = encodeURIComponent(mxUtils.getXml(node));
mxUtils.post(url, 'xml='+xmlString, function(req) {
  // Process server response using req of type mxXmlRequest
});
```

1.14 输入

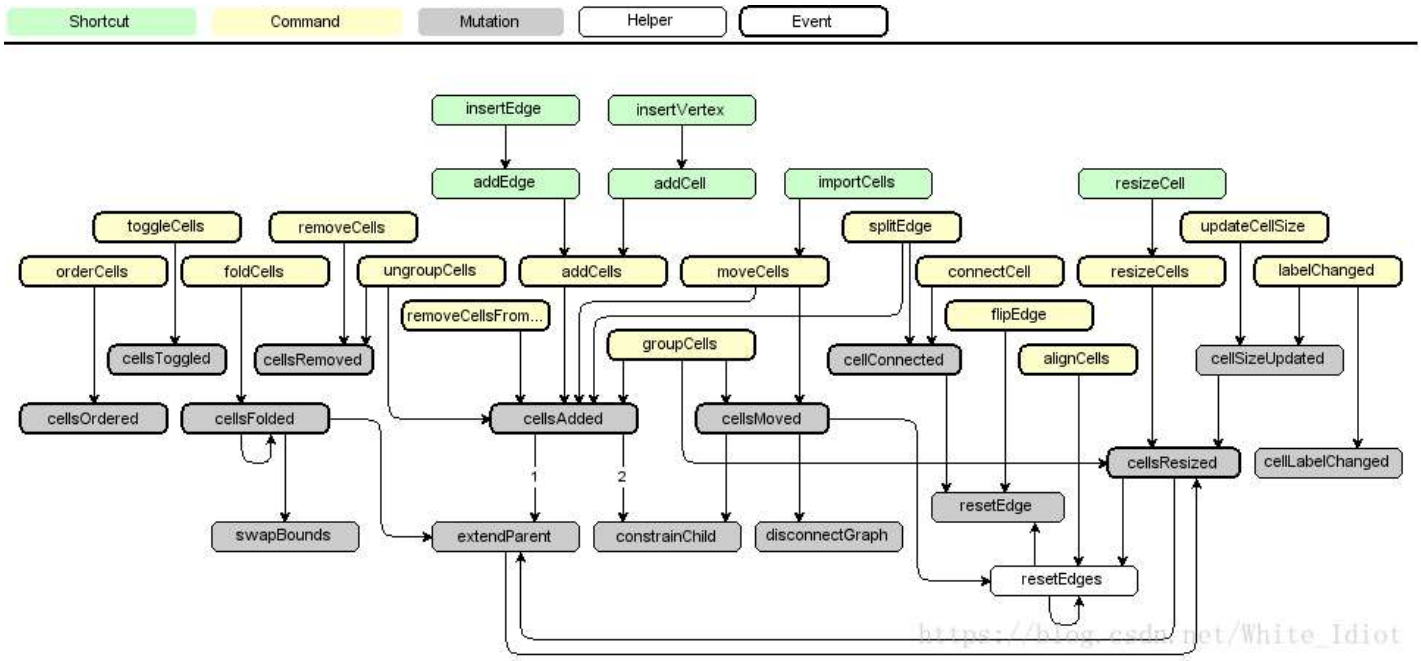
要将graph的XML表示加载到现有graph对象中，可以按如下方式使用mxUtils.load。url变量是生成XML字符串的Java servlet、PHP页面或HTTP处理程序的URL。

```
var xmlDoc = mxUtils.load(url).getXml();
var node = xmlDoc.documentElement;
var dec = new mxCodec(node.ownerDocument);
dec.decode(node, graph.getModel());
```

要创建使用单个请求加载客户端和图表的页面，请参阅后端中的部署示例。

1.15 功能依赖

mxGraph的功能依赖关系如图所示：



2. 构造

mxGraph的构造函数如下，可以了解到mxGraph的初始化过程：

```
function mxGraph(container, model, renderHint, stylesheet) {
    // 在原型被修改为保存一些监听器的情况下初始化变量
    // 这是可能的，因为无论传递给ctor的参数如何都要执行createHandlers调用。
    this.mouseListeners = null;

    // 将renderHint转换为dialect
    // 设置渲染cell的语言
    this.renderHint = renderHint;
    if (mxClient.IS_SVG) {
        this.dialect = mxConstants.DIALECT_SVG;
    } else if (renderHint === mxConstants.RENDERING_HINT_EXACT && mxClient.IS_VML) {
        this.dialect = mxConstants.DIALECT_VML;
    } else if (renderHint === mxConstants.RENDERING_HINT_FASTEST) {
        this.dialect = mxConstants.DIALECT_STRICTHTML;
    } else if (renderHint === mxConstants.RENDERING_HINT_FASTER) {
        this.dialect = mxConstants.DIALECT_PREFERHTML;
    } else { // default for VML
        this.dialect = mxConstants.DIALECT_MIXEDHTML;
    }

    // 初始化不需要容器的主要成员
    this.model = (model != null) ? model : new mxGraphModel();
    this.multiplicities = [];
    this.imageBundles = [];

    this.cellRenderer = this.createCellRenderer();
```

```
    this.cellRenderer = this.createCellRenderer();
    this.setSelectionModel(this.createSelectionModel());
    this.setStylesheet((stylesheet != null) ? stylesheet : this.createStylesheet());
    this.view = this.createGraphView();

    // 添加graph model监听器以更新视图
    this.graphModelChangeListener = mxUtils.bind(this, function (sender, evt) {
        this.graphModelChanged(evt.getProperty('edit').changes);
    });
    this.model.addListener(mxEvent.CHANGE, this.graphModelChangeListener);

    // 使用默认的禁用设置创建基本事件处理程序
    this.createHandlers();

    // 如果指定了container，则初始化显示
    if (container != null) {
        this.init(container);
    }

    this.view.revalidate();
}
```

mxGraph采用原型链方式继承自mxEventSource：

```
mxGraph.prototype = new mxEventSource();
mxGraph.prototype.constructor = mxGraph;
```

同时在mxGraph类加载时会加载所需的语言资源：

```
if (mxLoadResources) {
    mxResources.add(mxClient.basePath + '/resources/graph');
} else {
    mxClient.defaultBundles.push(mxClient.basePath + '/resources/graph');
}
```

在创建mxGraph实例时，如果传入了container则会调用init方法进行初始化并创建相应的数据结构：

```
/**
 * container - DOM结点用于包含graph
 */
mxGraph.prototype.init = function (container) {
    this.container = container;

    // 初始化就地编辑器
    this.cellEditor = this.createCellEditor();

    // 使用视图初始化容器
    this.view.init();

    // 更新当前图形的容器大小
    this.sizeDidChange();

    // 如果鼠标离开容器，则隐藏工具提示并重置工具提示计时器
    mxEvent.addListener(container, 'mouseleave', mxUtils.bind(this, function () {
        if (this.tooltipHandler != null) {
            this.tooltipHandler.hide();
        }
    }));

    // 自动释放内存

    if (mxClient.IS_IE) {
```

```
    if (mxClient.IS_IE) {
        mxEvent.addListener(window, 'unload', mxUtils.bind(this, function () {
            this.destroy();
        }));

        // 禁用文本的shift-click
        mxEvent.addListener(container, 'selectstart',
            mxUtils.bind(this, function (evt) {
                return this.isEditing() || (!this.isMouseDown && !mxEvent.isShiftDown(evt));
            })
        );
    }

    // 如果没有显示初始图形或没有定义形状标签，则在IE8标准模式下缺少最后一个形状和连接预览的解决方法
    if (document.documentMode == 8) {
        container.insertAdjacentHTML('beforeend', '<' + mxClient.VML_PREFIX + ':group' +
            ' style="DISPLAY: none;"></' + mxClient.VML_PREFIX + ':group>');
    }
};
```

3. 功能性

由于mxGraph定义了很多原型属性，这里不能一一列举，需要的时候可以查看 源码 。主要关注mxGraph功能性方面有哪些，以及结构如何。

3.1 句柄

mxGraph会在init方法调用之前创建几个句柄用于处理对应的事件，有如下几个句柄：tooltip、panning、connection、graph，不过目前都是关闭的：

```
mxGraph.prototype.createHandlers = function () {
    this.tooltipHandler = this.createTooltipHandler();
    this.tooltipHandler.setEnabled(false);

    this.selectionCellsHandler = this.createSelectionCellsHandler();

    this.connectionHandler = this.createConnectionHandler();
    this.connectionHandler.setEnabled(false);

    this.graphHandler = this.createGraphHandler();

    this.panningHandler = this.createPanningHandler();
    this.panningHandler.panningEnabled = false;

    this.popupMenuHandler = this.createPopupMenuHandler();
};
```

3.2 cell重叠

当cell重叠时，mxGraph需要额外进行一些操作记录重叠的cell并更新graph的显示，这样才能实现cell重叠层次的改变：

```
/**
 * cell - 被叠加的mxCell
 * overlay - 添加到cell上的mxCellOverlay
 */
mxGraph.prototype.addCellOverlay = function (cell, overlay) {
        // overlays是实例变量
    if (cell.overlays == null) {
        cell.overlays = [];
    }

    cell.overlays.push(overlay);
```

```
    var state = this.view.getState(cell);

    // 如果state存在，立即更新cell的重叠显示
    // state保存的是cell在graph中的所有状态
    if (state != null) {
        this.cellRenderer.redraw(state);
    }

    this.fireEvent(new mxEventObject(mxEvent.ADD_OVERLAY,
        'cell', cell, 'overlay', overlay));

    return overlay;
};
```

还有其他相关方法：getCellOverlays、removeCellOverlay、removeCellOverlays、clearCellOverlays，这里就不详述具体实现了。

3.3 就地编辑

就地编辑通过在graph中双击触发，可以在双击的地方创建一个文本输入框，具体实现如下：

```
/**
 * cell - 开始就地编辑的cell
 * evt - 可选的触发编辑的鼠标事件
 */
mxGraph.prototype.startEditingAtCell = function (cell, evt) {
    if (evt == null || !mxEvent.isMultiTouchEvent(evt)) {
        if (cell == null) {
            cell = this.getSelectionCell();

            // 选中的cell是否可以编辑
            if (cell != null && !this.isCellEditable(cell)) {
                cell = null;
            }
        }

        if (cell != null) {
            // 触发START_EDITING事件
            this.fireEvent(new mxEventObject(mxEvent.START_EDITING,
                'cell', cell, 'event', evt));
            // 开始编辑
            this.cellEditor.startEditing(cell, evt);
            // 触发EDITING_STARTED事件
            this.fireEvent(new mxEventObject(mxEvent.EDITING_STARTED,
                'cell', cell, 'event', evt));
        }
    }
};
```

还有其他相关方法：getEditingValue、stopEditing、labelChanged、cellLabelChanged、escape，这里就不详述具体实现了。

3.4 事件处理

事件处理方法的具体实现各异，不能一一详述，只需要知道如何触发，以及有什么效果即可，列举几个事件处理函数：

方法	描述
escape	处理ESC键事件
click	处理cell上的单击事件
dblClick	处理cell上的双击事件

方法	描述
tapAndHold	处理按住cell的事件

3.5 Cell样式

cell的样式处理有许多方法，列举几个graph处理cell的方法，这里不详述了：

方法	描述
getCellStyle	返回表示给定cell样式的键值对的数组
setCellStyle	设置指定cell的样式。 如果没有给出cell，则改变选中的cell
toggleCellStyle	以给定cell的样式切换给定键的布尔值，并将新值返回为0或1。如果未指定cell，则使用选中的cell
setCellStyleFlags	在指定cell的样式中设置或切换给定键的给定位

3.6 Cell的排列和方向

这里涉及处理cell的排列和方向的许多方法，列举一些就不详述了：

方法	描述
alignCells	使用可选参数作为坐标，根据给定的对齐方式垂直或水平对齐给定cell
flipEdge	在null（或空）和alternateEdgeStyle之间切换给定边的样式。事务正在进行时，此方法将触发mxEvent.FLIP_EDGE。返回被翻转的边
addImageBundle	添加指定的mxImageBundle
orderCells	将给定的cell移动到前面或后面。使用cellsOrdered执行更改。事务正在进行时，此方法将触发mxEvent.ORDER_CELLS

3.7 分组

graph的分组是一个非常强大的功能，列举一些方法：

方法	描述
groupCells	将cell添加到给定组中。使用cellsAdded，cellsMoved和cellsResized执行更改。事务正在进行时，此方法将触发mxEvent.GROUP_CELLS。返回新组。仅当给定cell数组中至少有一个条目时，才会创建组
getBoundsForGroup	返回用于给定组和子项的边界
createGroupCell	如果没有为group函数提供组cell，则钩子用于创建组cell以保存给定的mxCells数组

3.8 Cell克隆、插入和删除

graph的cell可以克隆、插入，也可以删除，列举一些常用的方法：

方法	描述
cloneCells	返回给定cell的克隆。克隆是使用mxGraphModel.cloneCells递归创建的。如果Edge的终端不在给定阵列中，则为相应的端分配终端点并移除终端
insertVertex	使用value作为用户对象并将给定坐标作为新vertex的mxGeometry，将新vertex添加到给定父mxCell中。id和style用于返回的新mxCell的各个属性
removeCells	如果includeEdges为true，则从graph中移除给定的cell，包括所有连接的edge。使用cellsRemoved执行更改。事务正在进行时，此方法将触发mxEvent.REMOVE_CELLS。删除的单元格作为数组返回

3.9 Cell的可见性

cell能设置它的可见性，用如下两个方法设置：

方法	描述
toggleCells	如果includeEdges为true，则设置指定cell和所有连接edge的可见状态。使用cellsToggled执行更改。事务正在进行时，此方法将触发mxEvent.TOGGLE_CELLS。返回可见状态已更改的cell
cellsToggled	设置指定cell的可见状态

3.10 折叠

graph的分组以及有些cell是可以折叠的，用如下方法实现：

方法	描述
foldCells	如果recurse为true，则设置指定cell和所有后代的折叠状态。使用cellsFolded执行更改。事务正在进行时，此方法将触发mxEvent.FOLD_CELLS。返回折叠状态已更改的cell
swapBounds	在执行交换之前，交换调用updateAlternateBounds的给定cell的几何中的替代边界和实际边界

3.11 Cell大小

列举一些改变cell大小的方法：

方法	描述
updateCellSize	使用cellSizeUpdated更新模型中给定cell的大小。事务正在进行时，此方法将触发mxEvent.UPDATE_CELL_SIZE。返回其大小已更新的单元格
getPreferredSizeForCell	返回给定mxCell的首选宽度和高度，作为mxRectangle。要实现最小宽度，要添加新样式
resizeCell	使用resizeCells设置给定cell的边界。返回传递给函数的cell
resizeChildCells	相对于cell的当前几何图形，针对给定的新几何图形调整给定cell的子cell的大小

3.12 Cell移动

列举一些移动cell的方法：

方法	描述
importCells	使用move方法克隆并将给定cell插入到图形中，并返回插入的cell。如果通过datatransfer插入cell，则使用此快捷方式
moveCells	移动或克隆指定的cell，并按给定的数量移动cell或克隆，将它们添加到可选的目标cell中。当鼠标被释放时，evt是鼠标事件。 使用cellsMoved执行更改。事务正在进行时，此方法将触发mxEvent.MOVE_CELLS。返回已移动的cell
translateCell	转换给定cell的几何图形，并将新的，已转换的几何图形作为原子更改存储在模型中

3.13 Cell连接及其约束

列举一些cell连接，和连接约束的方法：

方法	描述
getOutlineConstraint	返回用于连接到给定状态轮廓的约束

方法	描述
getAllConnectionConstraints	返回给定终端的所有mxConnectionConstraints的数组。 如果给定终端的形状是mxStencilShape，则返回相应mxStencil的约束
getConnectionPoint	返回绝对点列表中最近的点或相对终端的中心
connectCell	在事务正在进行时，使用cellConnected将给定边的指定端连接到给定终端并触发mxEvent.CONNECT_CELL。返回更新的边缘

3.14 追溯

列举一些追溯cell关系的方法：

方法	描述
getCurrentRoot	返回显示的单元格层次结构的当前根。这是view中mxGraphView.currentRoot的快捷方式
getTerminalForPort	返回用于给定端口的终端。此实现始终返回父单元格
enterGroup	使用给定的单元格作为显示的单元格层次结构的根。如果未指定单元格，则使用选择单元格。仅当isValidRoot返回true时才使用该单元格
home	使用模型的根作为显示的单元层次结构的根，并选择以前的根

3.15 graph显示

列举一些graph显示相关的方法：

方法	描述
getGraphBounds	返回可见图的边界。mxGraphView.getGraphBounds的快捷方式
getCellBounds	返回给定单元格的缩放，平移边界
refresh	清除从给定单元格开始的层次结构的所有单元格状态或状态，并验证图形。 这会触发刷新事件作为最后一步
snap	如果gridEnabled为true，则将给定的数值捕捉到网格
panGraph	按给定的量移动图表显示。这用于预览平移操作，使用mxGraphView.setTranslate设置视图的持久转换。触发mxEvent.PAN
zoomIn	通过zoomFactor放大图形
zoomOut	通过zoomFactor缩小图表
center	将图表置于容器中心

3.16 验证

列举一些验证性的方法：

方法	描述
validationAlert	在对话框中显示给定的验证错误。此实现使用mxUtils.alert
isEdgeValid	检查给定参数的getEdgeValidationError的返回值是否为null
validateGraph	通过验证给定单元格的每个后代或模型的根来验证图形。Context是一个包含完整验证运行的验证状态的对象。使用setCellWarning将验证错误附加到其单元格。在成功验证的情况下返回null，或者在验证失败的情况下返回字符串数组（警告）
getCellValidationError	检查在修改图形时不能强制执行的所有multiplicities，即所有需要至少1个边缘的多重性

3.17 graph外观

列举一些graph外观相关的方法：

方法	描述
setBackgroundImage	设置新的backgroundImage
getFoldingImage	返回用于显示指定单元状态的折叠状态的mxImage。这将为所有边返回null
getLabel	返回表示给定单元格标签的字符串或DOM节点。如果labelsVisible为true，则此实现使用convertValueToString。否则返回一个空字符串

3.18 graph行为

列举一些graph行为相关的方法：

方法	描述
isResizeContainer	设置resizeContainer
isCellLocked	如果无法移动，调整大小，弯曲，断开连接，编辑或选择给定单元格，则返回true。如果locked为false，则对于具有相对几何的所有顶点，此实现返回true
getCloneableCells	返回可在给定单元格数组中导出的单元格

3.19 获取Cell

列举一些graph行为相关的方法：

方法	描述
getDefaultParent	如果两者都为null，则返回defaultParent或mxGraphView.currentRoot或mxGraphModel.root的第一个子子项。此函数返回的值应该用作新单元格（也就是默认图层）的父级
getSwimlane	返回给定单元格的最近祖先，如果泳道本身是泳道，则返回泳道或给定单元格
getCellAt	返回与给定父级开始的单元层次结构中给定点（x，y）相交的最底部单元格。 如果给定位置与泳道的内容区域相交，这也将返回泳道。 如果不希望这样，那么如果返回的单元格是泳道，则可以使用hitsSwimlaneContent来确定该位置是在内容区域内还是在泳道的实际标题上

3.20 选中

列举一些跟选中相关的方法：

方法	描述
isCellSelected	如果选择了给定单元格，则返回true
isSelectionEmpty	如果选择为空，则返回true
getSelectionCells	返回所选mxCells的数组

3.21 选中状态

列举一些跟选中状态相关的方法：

方法	描述
createHandler	为给定的单元格状态创建一个新的处理程序。 此实现返回相应单元格的新mxEdgeHandler是边缘，否则返回mxVertexHandler

方法	描述
createVertexHandler	挂钩为给定的mxCellState创建新的mxVertexHandler
createEdgeHandler	挂钩为给定的mxCellState创建新的mxEdgeHandler

3.22 graph事件

列举一些graph事件相关的方法：

方法	描述
addMouseListener	向图形事件调度循环添加侦听器。监听器必须实现mouseDown，mouseMove和mouseUp方法，如mxMouseEvent类所示
updateMouseEvent	如果需要，设置给定wxMouseEvent的图形和graphY属性，并返回事件
getStateForTouchEvent	返回给定触摸事件的状态