



TWaver[®] HTML5

开发手册

Version 1.0

Mar 2012

Serva Software

info@servasoftware.com

<http://www.servasoftware.com>

PO Box 8143, Wichita Falls, Texas, USA 76307



For more information about Serva Software and TWaver please visit the web site at:

<http://www.servasoftware.com>

Or send e-mail to:

info@servasoftware.com

Mar, 2012

Notice:

This document contains proprietary information of Serva Software. Possession and use of this document shall be strictly in accordance with a license agreement between the user and Serva Software, and receipt or possession of this document does not convey any rights to reproduce or disclose its contents, or to manufacture, use, or sell anything it may describe. It may not be reproduced, disclosed, or used by others without specific written authorization of Serva Software.

TWaver, servasoft, Serva Software and the logo are registered trademarks of Serva Software. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S.A. and other countries. Other company, brand, or product names are trademarks or registered trademarks of their respective holders. The information contained in this document is subject to change without notice at the discretion of Serva Software.

Copyright © 2012 Serva Software LLC
All Rights Reserved

Table of Contents

- 入门
 - 约定俗成
 - TWaver HTML开发环境
 - TWaver HTML5快速上手
- 基础
 - TWaver HTML5设计模式与结构
 - TWaver HTML5数据模型
 - TWaver HTML5数据元素
 - TWaver HTML5数据管理容器
 - TWaver HTML5视图组件
 - Network组件介绍
 - Tree组件介绍
 - Table组件介绍
 - 数据序列化
- 数据元素
 - 基本数据元素
 - 告警元素
 - 图层元素
 - 拓扑元素
 - twaver.Node
 - twaver.Link
 - 连线类型
 - 连线绑定
 - twaver.Follower
 - twaver.Grid
 - twaver.Group
 - twaver.SubNetwork
 - twaver.ShapeNode
- 数据容器
 - 数据容器之事件机制
 - 快速查找 - QuickFinder
 - 选中模型 - SelectionModel
- Network组件
 - Network层次结构
 - Network过滤器的使用
 - Network样式规则函数
 - Network界面交互
- Tree组件
 - 树节点样式
 - 树的层次与排序
 - 树的勾选模式
- Table组件
 - 表格列的设置
 - 表格数据排序

- Chart组件
 - Chart Value & Chart Values
 - BarChart
 - LineChart
 - PieChart
 - DialChart
 - BubbleChart
 - RadarChart
- 告警的使用
 - 告警级别
 - 告警状态与统计
 - 告警呈现
- 附录
 - 网元样式表

入门

本章介绍TWaver HTML5入门开发需要注意的事项，从HTML + JS编程环境如何搭建，创建新的Web工程，到各个TWaver组件的实例介绍，让用户可以快速利用TWaver HTML5开发图形化界面。

本节内容：

- [约定俗成](#)
- [TWaver HTML开发环境](#)
- [TWaver HTML5快速上手](#)

约定俗成

初次使用TWaver HTML5，为了更好的理解本文档，这里列举一些约定俗成的说法以及对比与TWaver Java版本的用语：

MVC - 模型-视图-控制器模式

数据模型：对应MVC中的Model，指Data, DataBox, ElementBox, AlarmBox, LayerBox, Element...

视图：对应MVC中的View，指ElementUI, Network, Tree, Table...

数据元素：Data, Element, Node...

数据容器：特指存放数据元素的容器，DataBox, ElementBox, AlarmBox, LayerBox

类名与TWaver Java版本的对比

TWaver HTML5	TWaver Java	说明
ElementBox	TDataBox	网元容器
AlarmBox	AlarmModel	告警容器
LayerBox	LayerModel	图层容器
Network	TNetwork	拓组件件
Tree	TTree	树图组件
Table	TElementTable	表格组件

网元属性设置对比TWaver Java版本

TWaver HTML5	TWaver Java	说明
node.setStyle	node.putClientProperty	样式属性
node.setClient	node.putUserProperty	用户属性

Network中过滤器命名与TWaver Java版本对比：

TWaver HTML5	TWaver Java	说明
getAlarmLabel	alarmLabelGenerator	告警冒泡标签
isVisible	visibleFilter	可见过滤器
isMovable	movableFilter	可移动过滤器
isEditable	elementLabelEditableFilter	可编辑过滤器
getLabel	elementLabelGenerator	网元标签文本生成器
getToolTip	elementToolTipTextGenerator	提示文本生成器
getInnerColor	elementBodyColorGenerator	网元体颜色渲染

getOuterColor	elementOutlineColorGenerator	网元边框颜色渲染
getSelectColor	elementSelectColorGenerator	选中颜色生成器
getAlarmFillColor	alarmColorGenerator	告警冒泡颜色
getAlarmLabel	alarmLabelGenerator	告警冒泡文本

TWaver HTML开发环境

本章介绍之前，首先你需要有twaver.js等库文件，可以在 www.servasoftware.com 网站申请试用版本，商业用途开发的用户需要购买TWaver HTML5开发版许可。

开发工具

开发Web项目的工具很多，从最简单的文本编辑器到Web集成IDE，没有一种权威的推荐，相比Java，.NET之类的静态编程语言，JavaScript,HTML,CSS的开发更适合用简单的文本编辑器，因此本文档将不推荐专门的开发工具。

支持的浏览器

TWaver HTML5使用JavaScript语言开发，可运行在多种现代浏览器：IE9+, Firefox, Chrome, Safari, Opera。请原谅我没有列出具体的版本号，现在浏览器更新换代太快，而TWaver HTML5面向的是现代和未来的浏览器，所以忘掉那些落后的老浏览器吧。

参考资料

[HTML5 at W3C](#) - 标准和学习资料

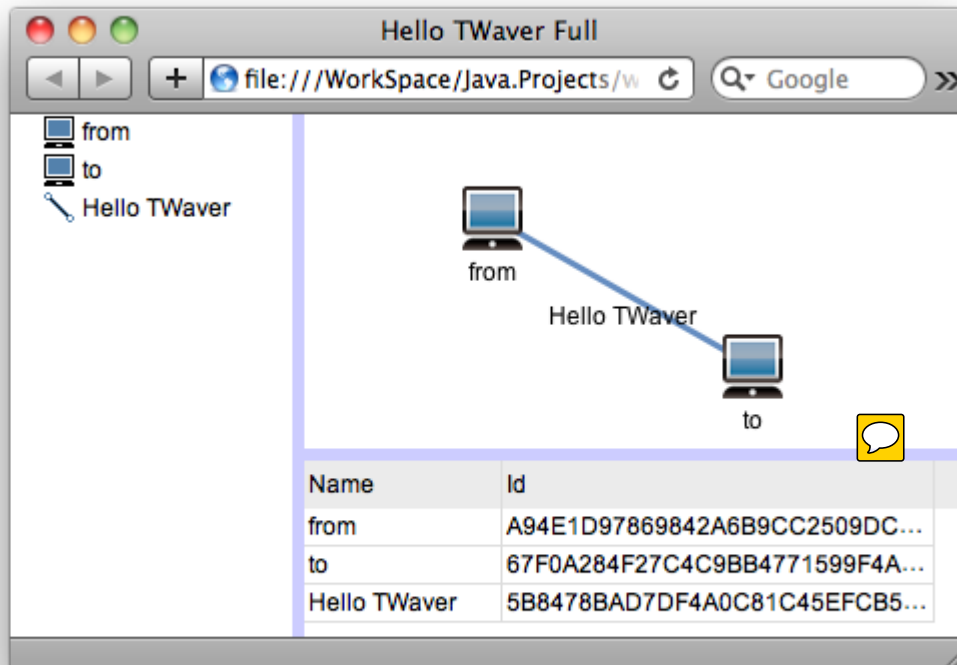
[HTML5 Labs](#) - web标准的早期原型（比如 IndexedDB, FileAPI 等等）

[ScriptJunkie.com](#) - web开发文章和信息

[CanIUse.com](#) - 各个浏览器对HTML5, CSS3以及其他技术支持的详细信息

TWaver HTML5快速上手

下面将通过实例介绍TWaver HTML5的使用，本例中包括树，拓扑图，表格三种组件，最终效果如下：



下面我将分步骤讲解

创建html文件，引入相关twaver.js，注意html doctype的设置

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello TWaver Full</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

这里需要注意文件头信息的设置"<!DOCTYPE html>"，表示文档类型，部分浏览器下必须设置这个头信息，才能创建HTML5中的Canvas元素

创建拓扑图组件

下面来创建一个拓扑图，实现简单的节点连线，了解TWaver HTML5的大体编程模式

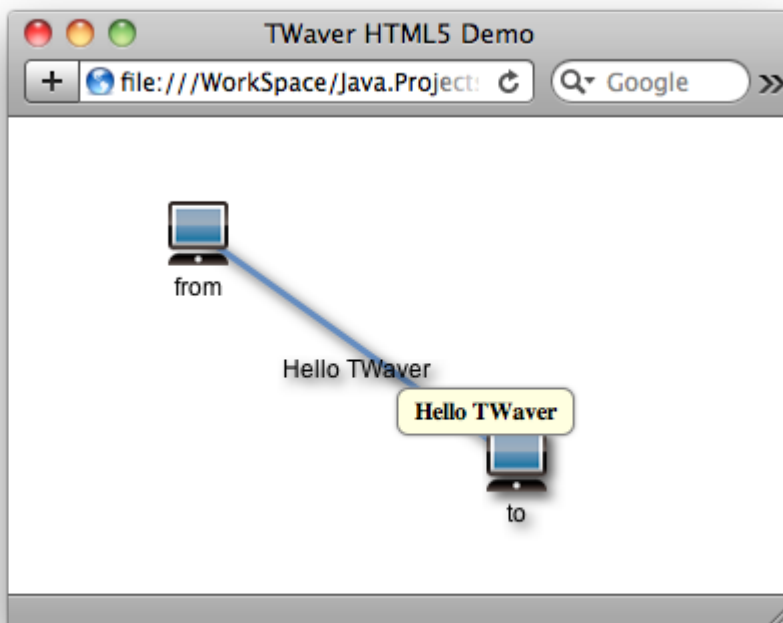
```
function init() {
```

```
var network = new twaver.network.Network();

var box = network.getElementBox();
var node = new twaver.Node();
node.setName("from");
node.setLocation(100, 100);
box.add(node);
var node2 = new twaver.Node();
node2.setName("to");
node2.setLocation(300, 300);
box.add(node2);
var link = new twaver.Link(node, node2);
link.setName("Hello TWaver");
link.setToolTip("<b>Hello TWaver</b>");
box.add(link);

var networkDom = network.getView();
networkDom.style.width = "100%";
networkDom.style.height = "100%";
document.body.appendChild(networkDom);
}
```

在浏览器中运行



增加其他组件，使用TWaver的劈分面板布局

```
function init() {
    var box = new twaver.ElementBox();
    var node = new twaver.Node();
    node.setName("from");
    node.setLocation(100, 100);
    box.add(node);
    var node2 = new twaver.Node();
    node2.setName("to");
    node2.setLocation(300, 300);
```

```

        box.add(node2);
        var link = new twaver.Link(node, node2);
        link.setName("Hello TWaver");
        link.setToolTip("<b>Hello TWaver</b>");
        box.add(link);

        var network = new twaver.network.Network(box);
        var tree = new twaver.controls.Tree(box);
        var table = new twaver.controls.Table(box);
        var tablePane = new twaver.controls.TablePane(table);
        createColumn(table, 'Name', 'name', 'accessor', 'string');
        createColumn(table, 'Id', 'id', 'accessor', 'string');

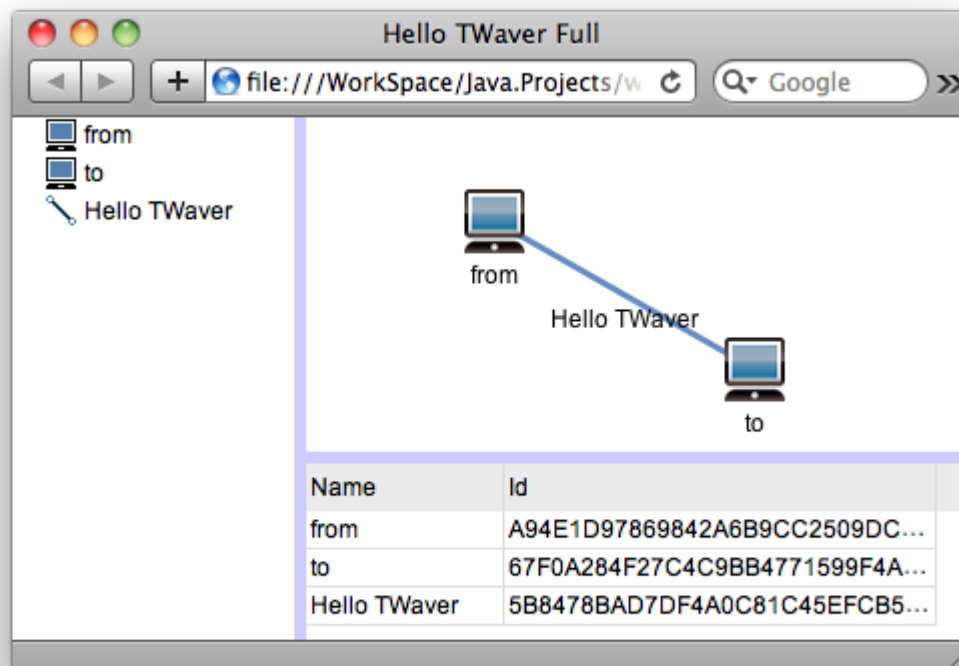
        var rightSplit = new twaver.controls.SplitPane(network, tablePane,
            'vertical', 0.7);
        var mainSplitPane = new twaver.controls.SplitPane(tree, rightSplit,
            'horizontal', 0.3);

        var networkDom = mainSplitPane.getView();
        networkDom.style.width = "100%";
        networkDom.style.height = "100%";
        document.body.appendChild(networkDom);
        network.getView().style.backgroundColor = "#f3f3f3";
        network.getView().style.cursor = "hand";

        window.onresize = function() {
            mainSplitPane.invalidate();
        };
    }
    function createColumn(table, name, propertyName, propertyType, valueType) {
        var column = new twaver.Column(name);
        column.setName(name);
        column.setPropertyName(propertyName);
        column.setPropertyType(propertyType);
        if (valueType)
            column.setValueType(valueType);
        table.getColumnBox().add(column);
        return column;
    }
}

```

看看最终的运行界面：



基础

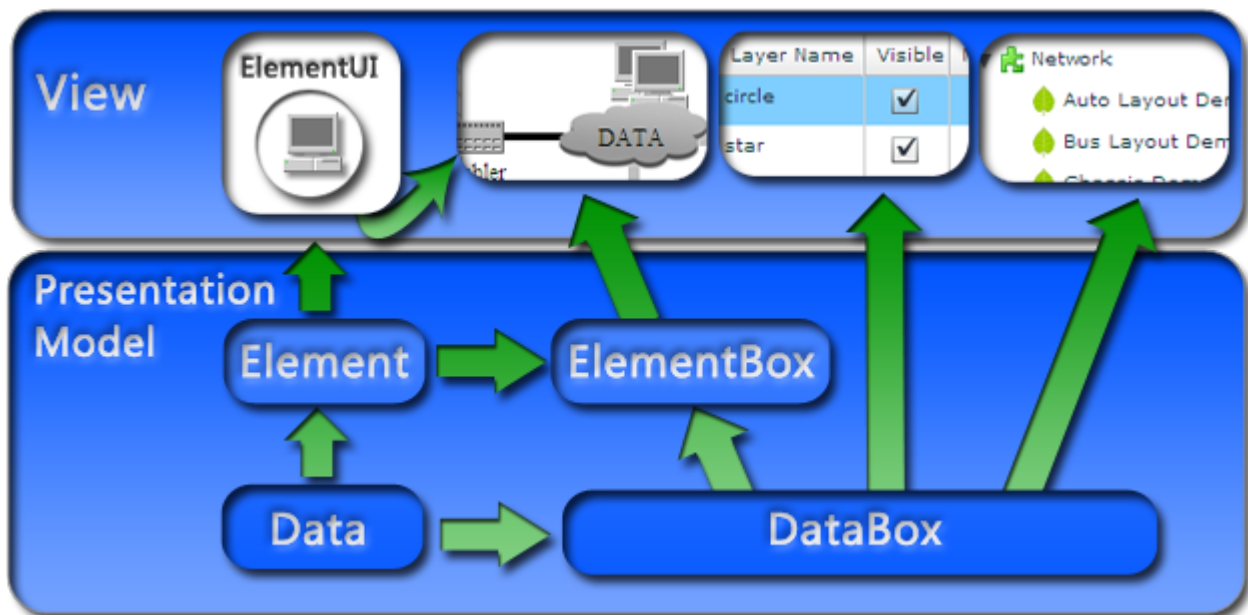
本章将介绍TWaver HTML5图形组件的设计模式及其组成结构，并对数据模型和各种视图组件做简单的介绍。阅读本章可以基本了解TWaver HTML5的原理和使用。

- [TWaver HTML5设计模式与结构](#)
- [TWaver HTML5数据模型](#)
 - [TWaver HTML5数据元素](#)
 - [TWaver HTML5数据管理容器](#)
- [TWaver HTML5视图组件](#)
 - [Network组件介绍](#)
 - [Tree组件介绍](#)
 - [Table组件介绍](#)
- [数据序列化](#)

TWaver HTML5设计模式与结构

TWaver HTML5采用基本的模型视图分离的设计思想，通过MV的组合嵌套，实现了以twaver.Data为基本数据元素，twaver.DataBox为基本数据容器的客户端数据模型，以及基本图形twaver.network.ElementUI和数据容器组件（Network, Tree, Table...）的视图系统，三者共同构成了TWaver HTML5的架构体系。

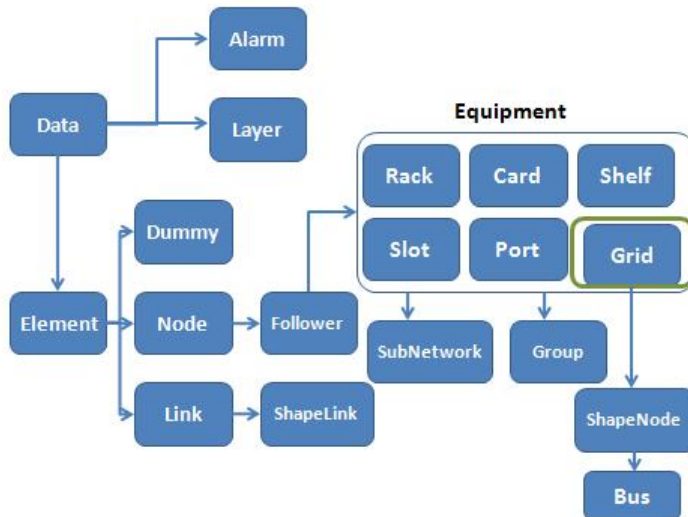
TWaver HTML设计模式



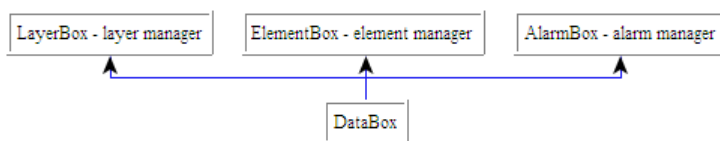
TWaver HTML5数据模型

TWaverHTML5的基本数据元素是twaver.Data，基本数据容器定义为twaver.DataBox，基于这两类基本元素，TWaverHTML5预定义了一系列业务对象，视图网元和管理容器，例如告警（twaver.Alarm）和告警容器（twaver.AlarmBox），图层（twaver.Layer）与图层管理容器（twaver.LayerBox），拓扑网元（twaver.Element）和拓扑管理容器（twaver.ElementBox）.....

数据元素结构图



数据管理容器结构图



其中拓扑管理容器（twaver.ElementBox）整合了其他几种容器，提供了丰富的拓扑元素（Dummy, Node, Link, Bus, ShapeNode, ShapeLink, Follower, Rack, Shelf, Slot, Card, Port, Grid, Group, SubNetwork.....），为网管界面开发提供了强大的设计模型和业务功能的基础支持。

- [TWaver HTML5数据元素](#)
- [TWaver HTML5数据管理容器](#)

TWaver HTML5数据元素

TWaverHTML5以twaver.Data为最基本的数据单元，扩展定义了一系列具有图形和业务意义的数据类型，包括Alarm, Layer, Element...

- **twaver.Data**

Data是TWaverHTML5的数据元素基类，定义了id, name, icon, toolTip, parent, children等基本属性，对事件派发做了封装。

Data包含一个twaver.EventDispatcher实例对象，这使它具有事件派发和监听的功能，可以通过调用下面的方法派发事件或者添加实现监听器：

```
firePropertyChange: function (property, oldValue, newValue)
addPropertyChangeListener: function (listener, scope, ahead)
removePropertyChangeListener: function (listener)
onPropertyChanged: function(listener)
```

此外，Data中还定义其他功能函数

```
getChildren: function ()
getChildrenSize: function ()
toChildren: function (matchFunction, scope)
addChild: function (child, index)
removeChild: function (child)
getChildAt: function (index)
clearChildren: function ()
getParent: function ()
setParent: function (parent)
hasChildren: function ()
isRelatedTo: function (data)
isParentOf: function (data)
isDescendantOf: function (data)
```

下面分别介绍各个实现类

- **twaver.Layer**

图层，用于TWaver的图层管理，实现类是twaver.Layer，它有三个特殊属性：visible, editable, movable，分别表示图层是否可见，是否可编辑，是否可移动。

TWaverHTML5中的层次关系由LayerBox来管理，默认的层次顺序由父子关系和先后顺序决定，在拓扑图中，每个Element通过设置layerId与某个layer相关联以控制网元的显示层次。

- **twaver.Alarm**

告警，用来表示网管系统中设备故障或者网络异常的数据模型，基本实现类是Alarm。告警与Element相关联，用以反映网元的告警状态，Alarm中定义了级别，是否已清除，是否已确认以及相关联的网元编号。

TWaverHTML5预定义了六种告警级别，告警级别的value属性可表示告警的严重程度，默认value值越大，告警越严重。

Severity	Letter	Value	Color
CRITICAL	C	500	Red

MAJOR	M	400	Orange
MINOR	m	300	Yellow
WARNING	W	200	Cyan
INDETERMINATE	N	100	Purple
CLEARED	R	0	Green

TWaver中告警使用AlarmBox进行管理，告警与网元通过AlarmBox来相关联，两者不直接引用，与网元直接引用的是AlarmState，用来反映新发告警的级别和数量

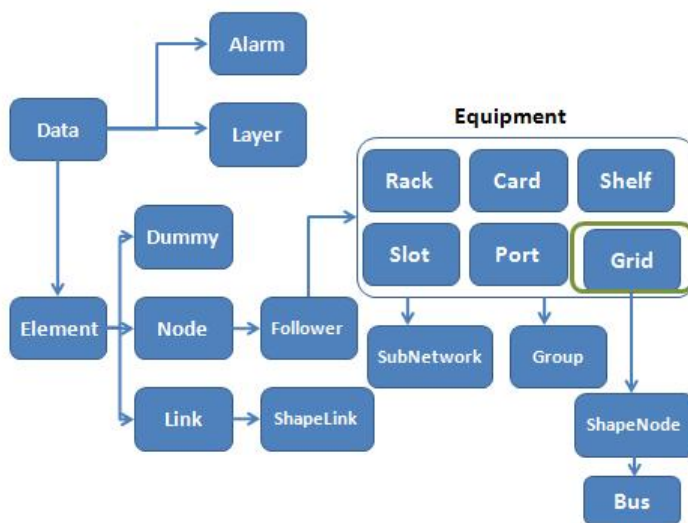
• twaver.Element

IElement是TWaver中最重要的数据元素，Element是其实现类，用于表示拓扑图中的网元对象，如节点，连线，子网，分组，板卡.....

TWaver预定义了丰富的网元类型（Dummy, Node, Link, Bus, ShapeNode, ShapeLink, Follower, Rack, Shelf, Slot, Card, Port, Grid, Group, SubNetwork.....），每一种网元对应一个ElementUI类，对应网元在拓扑图中的呈现组件类型，两者构成一个模型与视图分离的模型结构。

通过设置网元的属性和样式可以表现出丰富的呈现效果和特性，用户也可以扩展这些预定义的Element，或者定制自己的ElementUI，已应对特殊的业务呈现需求。

Element数据可以用ElementBox管理，ElementBox可以驱动twaver.Network, twaver.Tree, twaver.Table等多种视图



Dummy

在拓扑图中不可见，可在树，表格中显示，通常用来表示无拓扑意义的逻辑分组

Node

表示拓扑图中的一个节点，是其他节点的基类

Link

表示连线，是其他连线类型的基类

Follower

表示跟随者，可以附着在另一个Node（称之为宿主节点）上，宿主节点移动，Follower也跟着移动

Bus

继承于ShapeNode，是一种布局类型节点，可以与连接在它上面的节点们排布出总线布局那样的效果

ShapeNode

由一系列控制点决定形状，可以表现丰富的形态

ShapeLink

继承于Link，与Link不同，其走向有一系列控制点决定，可以定制出特殊的连线布局

Grid

在拓扑图上表现为网格，可以指定行列数，是Rack, Shelf, Slot, Card, Port的基类，可以用来表示设备面板

Group

表示分组，包含孩子网元，可以展开合并，孩子的位置和范围决定Group展开后的位置和范围

SubNetwork

子网在拓扑图中有重要意义，拓扑图通常并不会一次显示所有的网元，而只显示当前子网中的元素，通过切换子网和数据的延时加载可以解决大数据量的问题

TWaver HTML5数据管理容器

数据管理容器，顾名思义就是用来管理数据的容器，TWaverHTML5中的DataBox就是用来管理所有Data数据的容器，在TWaverHTML5的设计模式中担当者Model或者ViewModel的重要角色，一个DataBox可以驱动多个视图，DataBox中数据的变化都能够自动的反映到其关联的所有视图组件上。TWaverHTML5中的DataBox支持的视图包括：twaver.controls.Tree, twaver.controls.Table，此外由DataBox扩展而来的ElementBox还有专门的视图组件twaver.network.Network。

DataBox

DataBox包含一个EventDispatcher实例属性，能监听容器变化以及容器内数据的变化，添加监听可以使用下面的相关函数

```
addDataBoxChangeListener: function (listener, scope, ahead)
removeDataBoxChangeListener: function (listener)
addDataPropertyChangeListener: function (listener, scope, ahead)
removeDataPropertyChangeListener: function (listener)
addHierarchyChangeListener: function (listener, scope, ahead)
removeHierarchyChangeListener: function (listener)
```

一个容器，自然要提供数据管理的方法，DataBox中定义了下面的方法进行数据增删操作

```
add: function (data, index)
remove: function (data)
clear: function ()
```

对数据的遍历查找也提供了简便的方法

```
getSiblings: function (data)
getRoots: function ()
getDats: function ()
getDataAt: function (index)
toDats: function (matchFunction, scope)
forEach: function (f, scope)
forEachReverse: function (f, scope)
forEachByDepthFirst: function (callbackFunction, data, scope)
forEachByBreadthFirst: function (callbackFunction, data, scope)
```

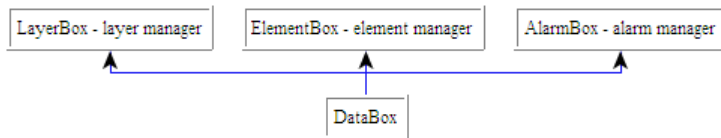
DataBox还管理着数据的层次模型，提供了下面的方法方便数据的移动和插入操作

```
moveTo: function (data, newIndex)
moveUp: function (data)
moveDown: function (data)
moveToTop: function (data)
moveToBottom: function (data)
```

此外DataBox对数据元素的选中机制做了封装，实现了数据的选中模型（SelectionModel），并提供了便捷操作的方法

```
getSelectionModel: function ()
```

TWaverHTML5中定义了LayerBox, AlarmBox, ElementBox分别用于图层管理，告警管理和网元管理，其中前两种容器都是为ElementBox服务的，用于管理网元的图层和告警信息



LayerBox

LayerBox是管理图层的容器，需要与一个ElementBox关联，其中定义了添加删除层的方法，还预定义了一个默认层

```
twaver.LayerBox = function (elementBox)
getElementBox: function ()
getDefaultLayer: function ()
getLayerByElement: function (element)
```

h3. AlarmBox

AlarmBox是管理告警的容器，首先需要与一个ElementBox相关联

```
twaver.AlarmBox = function (elementBox)
getElementBox: function ()
```

AlarmBox还定义了AlarmElementMapping用来定义网元与告警之间的对应关系，用户可以定制这一关系，实现一个告警影响多个网元的业务需求

```
getAlarmElementMapping: function ()
setAlarmElementMapping: function (alarmElementMapping)
```

此外还提供了下面的属性

```
//网元在ElementBox中被删除时，告警是否从AlarmBox中删除
//告警设置为清除级别时，是否自动删除告警
isRemoveAlarmWhenAlarmIsCleared: function ()
setRemoveAlarmWhenAlarmIsCleared: function (removeAlarmWhenAlarmIsCleared)
```

h3. ElementBox

ElementBox中包含告警容器，图层容器，是管理网元数据的容器，TWaverFlex中为ElementBox定义了专有的视图组件，用于图形化的显示网元的拓扑关系。

ElementBox中封装了很多功能，是TWaverHTML5中最重要和最常用的类，我们将在以后的章节中详细介绍。

ElementBox HelloWorld 示例

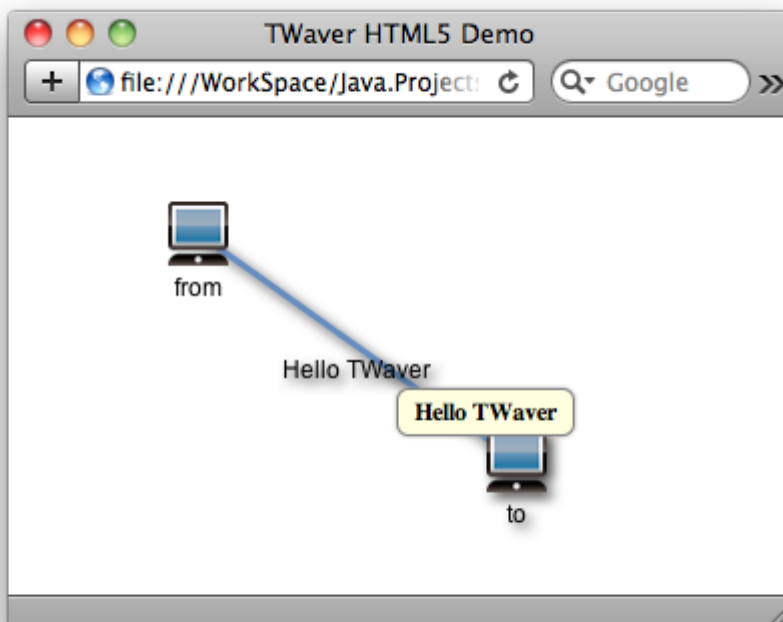
```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Hello TWaver Full</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init() {
      var network = new twaver.network.Network();

      var box = network.getElementBox();
      var node = new twaver.Node();
      node.setName("from");
      node.setLocation(100, 100);
      box.add(node);
      var node2 = new twaver.Node();
      node2.setName("to");
      node2.setLocation(300, 300);
      box.add(node2);
      var link = new twaver.Link(node, node2);
      link.setName("Hello TWaver");
      link.setToolTip("<b>Hello TWaver</b>");
      box.add(link);

      var networkDom = network.getView();
      networkDom.style.width = "100%";
      networkDom.style.height = "100%";
      document.body.appendChild(networkDom);
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

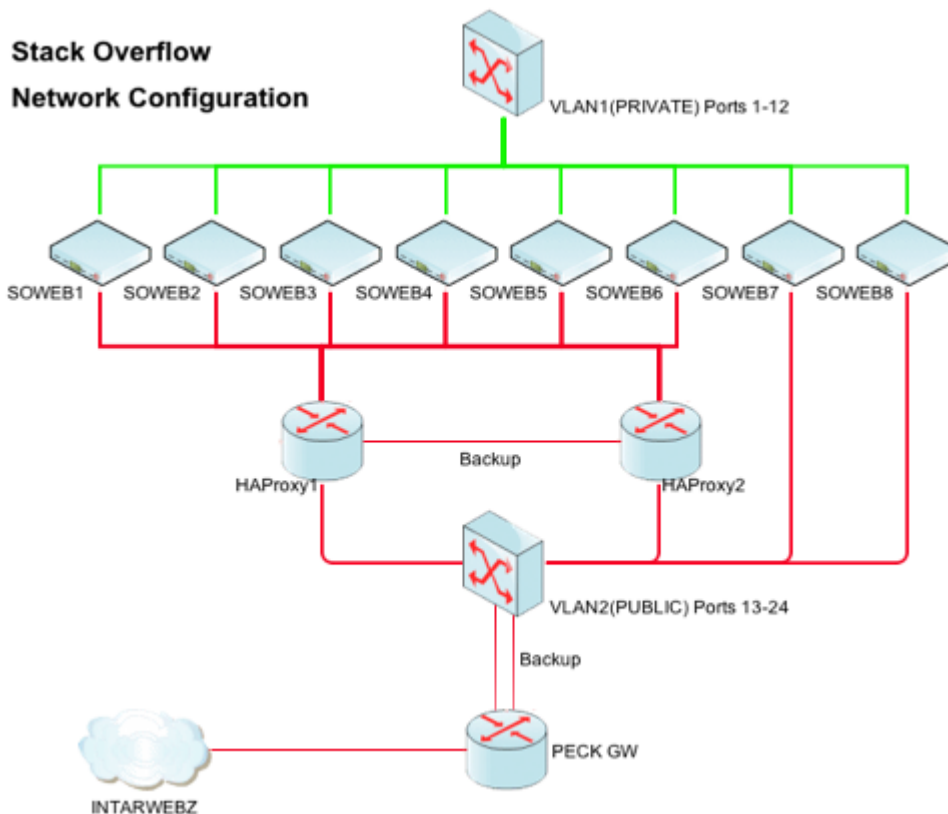
运行界面



TWaver HTML5视图组件

TWaverHTML5中定义了一些视图组件，如Network, Table, Tree...能通过DataBox驱动的成分，其中Network是展示拓扑图的图形化组件，Table, Tree, Chart为通用组件

拓扑图组件



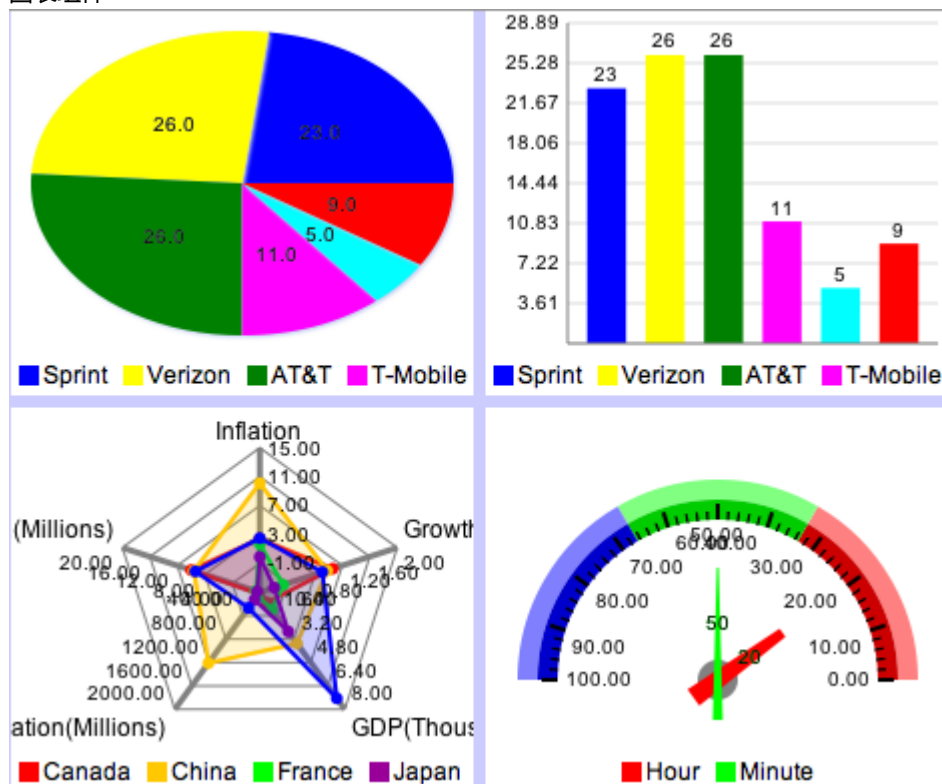
表格组件

Mapping ID	Alarm Severity	Acked	Cleared	Raised Time
1	Indeterminate	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07
2	Warning	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07
3	Minor	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07
4	Major	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07
5	Critical	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07
6	Indeterminate	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2011-11-04 11:38:07

树图组件

- Alarm Demos
 - Alarm Statistics Demo
 - Alarm Mapping Demo
 - Alarm Propagation Demo
- Network Demos
 - Topology Demos
 - PSTN Demo
 - Bundle Demo
 - Auto Layout Demo
 - Spring Layout Demo
 - States Map Demo
 - Bus Layout Demo
 - Layer Vector Demo
 - Success Story Demo

图表组件



- [Network组件介绍](#)
- [Tree组件介绍](#)
- [Table组件介绍](#)

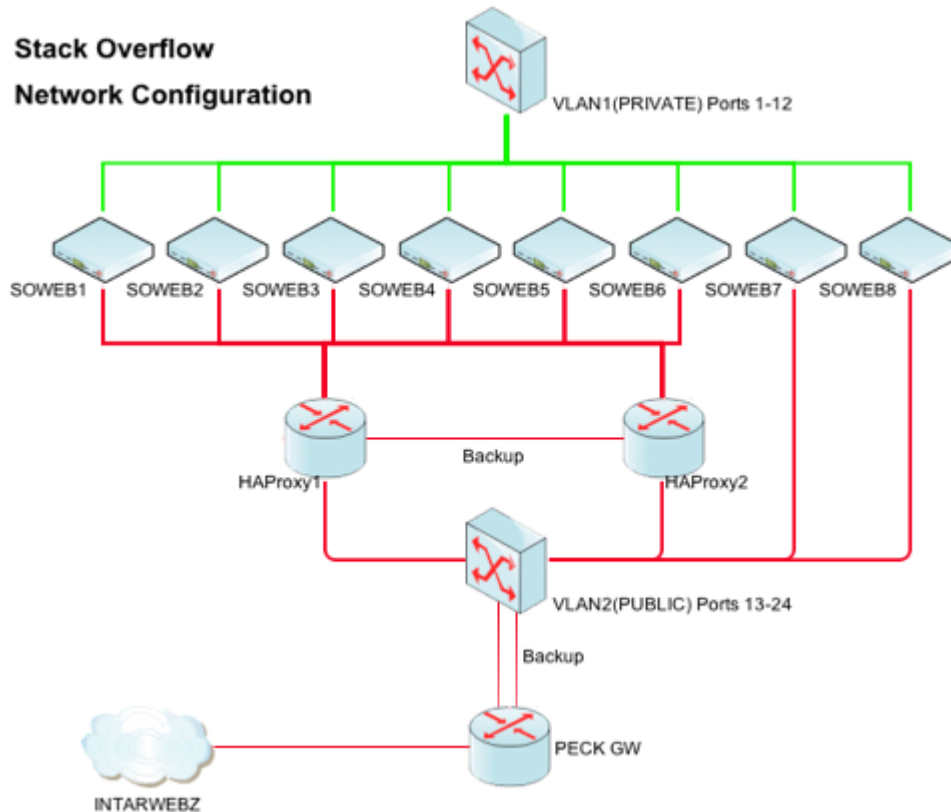
Network组件介绍

Network组件概要

Network组件用于图形化呈现网元信息，具有网元过滤，子网切换，背景图支持，缩放查看，多层机制，Attachment组件显示等功能，还提供了丰富的鼠标交互和动画效果，是网管系统中最直观、最美观的展示平台。此外Network还提供了上百个函数和属性，用户可以灵活的配置和扩展，以满足特殊的应用需求。

下面先展示一张示例截图，简要介绍Network中几个常用的功能，详细功能将在以后的章节介绍。

利用Network组件可以轻易的做出下面拓扑图效果



Network的组件模型和层次结构

Network使用HTML的DIV元素类型，其中包含各种图形元素，比如普通的<div>，元素，HTML5特有的<canvas>，<video>元素，甚至还可以嵌入其他浏览器支持的元素，比如：flash, SVG等等，此外Network组件还具有多层结构，有置顶层，背景层。

network 组件结构:

```
Network
-> rootDiv //根容器面板
-> topDiv //顶层面板
-> attachmentDiv //附件面板
-> layerDiv //网元视图面板，用于放置ElementUI,
-> layer n //ElementUI的层次关系由ElementBox.layerBox来管理
-> layer ...
-> layer 0
-> bottomDiv //底层容器面板
```



```
-> backgroundDiv //背景层面板
```

拓扑图中的网元组件

Network中每个网元对应一个ElementUI的视图组件，通过操作网元的数据模型（Element）可以反映到这个视图组件的呈现效果，Network中可以通过下面的方法得到网元对应的这个组件。

```
getElementUI : function(element)
```

Attachment组件

twaver.network.ui.Attachment，拓扑图中的网元不只能关联ElementUI组件，还可以绑定多个附件（Attachment）组件来表现属性，如网元标签，告警冒泡都是通过附件机制实现的，用户可以给网元定义新的附件以表现网元的属性

Attachment是一种视图组件，可以挂载在网元上，显示与拓扑图组件中，attachment可以置顶显示，通过设置showInAttachmentDiv属性可以控制其是添加到ElementUI还是独立加入到拓扑图中的附件层画布（attachmentDiv）中。

```
//Attachment构造函数
twaver.network.Attachment = function (elementUI, showInAttachmentDiv)

twaver.network.Network#
//network中的附件层画布，附件组件可以添加到ElementUI组件下，
//也可以与ElementUI组件分离，直接添加在拓扑图的附件层画布中，这样可以保证附件置顶显示
getAttachmentDiv: function ()

twaver.network.ui.ElementUI
//ElementUI中可以获取网元附件的集合
getAttachments: function ()
```

• Network常用功能

切换交互模式

Network中切换交互模式，能实现不同模式下不同的鼠标交互效果，最常用到的是选择模式（也叫默认模式）和编辑模式，用户也可以通过实现Interaction接口，处理鼠标键盘事件，以定制自己的交互模式，这些Interaction可以叠加使用，各种交互可以灵活编制在一起。

```
//切换到默认交互模式
setDefaultInteractions: function (lazyMode)
//切换到编辑模式
setEditInteractions: function (lazyMode)
//切换到创建节点交互模式
setCreateElementInteractions: function (type)
//切换到创建连线的交互模式
setCreateLinkInteractions: function (type)
//切换到创建ShapeNode交互模式
setCreateShapeNodeInteractions: function (type)
```

过滤器的使用

```
network.visibleFunction = function(node){  
    return node.getChildrenCount() > 0;  
};
```

列举部分属性

Network有上百个属性，这里列举一部分

```
getCurrentSubNetwork: function ()  
//设置当前子网  
setCurrentSubNetwork: function (currentSubNetwork, animate, finishFunction)  
  
//进入上级子网  
upSubNetwork: function (animate, finishFunction)  
  
//添加/删除交互（鼠标键盘）监听器  
addInteractionListener = function (listener, scope, ahead)  
removeInteractionListener = function (listener)  
  
//刷新网元视图  
invalidateElementUI: function (element, checkAttachments)  
invalidateElementUIs: function (checkAttachments)  
  
//获得网元视图  
getElementUI: function (element)  
  
//缩放操作  
getZoom = function ()  
setZoom = function (value, animate)  
zoomIn = function (animate)  
zoomOut = function (animate)  
zoomReset = function (animate)  
  
//获得拓扑图组件中的面板容器  
getRootDiv: function ()  
getTopDiv: function ()  
getAttachmentDiv: function ()  
getLayerDiv: function ()  
getBottomDiv: function ()  
getBackgroundDiv: function ()  
//获取拓扑图的DIV元素  
getView: function ()
```

Tree组件介绍

twaver.controls.Tree是一种视图组件，也称为树图组件，它可以与任意DataBox数据容器（LayerBox, AlarmBox, ElementBox...）绑定，展现其数据的层次关系，数据容器的层次关系由元素的父子关系决定，树图组件有自己的选中模型，也可以使用DataBox的选中模型，这样可以达到同步选中的效果，TWaver的树图组件最大特点是高效，采用了局部渲染机制，可以承载数十万的数量级。

树图组件的使用非常简单，只要关联上DataBox类型的数据容器，就可以展示这个容器中的层次结构

过滤器的使用

```
tree.setVisibleFunction(function(data){
    return data.getName().length < 10;
});
```

示例：

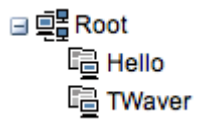
```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var tree = new twaver.controls.Tree();

      var box = tree.getDataBox();
      var root = new twaver.Data();
      root.setIcon("group_icon");
      root.setName('Root');
      box.add(root);
      var data = new twaver.Data();
      data.setParent(root);
      data.setName("Hello");
      box.add(data);
      data = new twaver.Data();
      data.setName("TWaver");
      data.setParent(root);
      box.add(data);

      var treeDom = tree.getView();
      treeDom.style.width = "100%";
      treeDom.style.height = "100%";
      document.body.appendChild(treeDom);

      tree.expandAll();
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

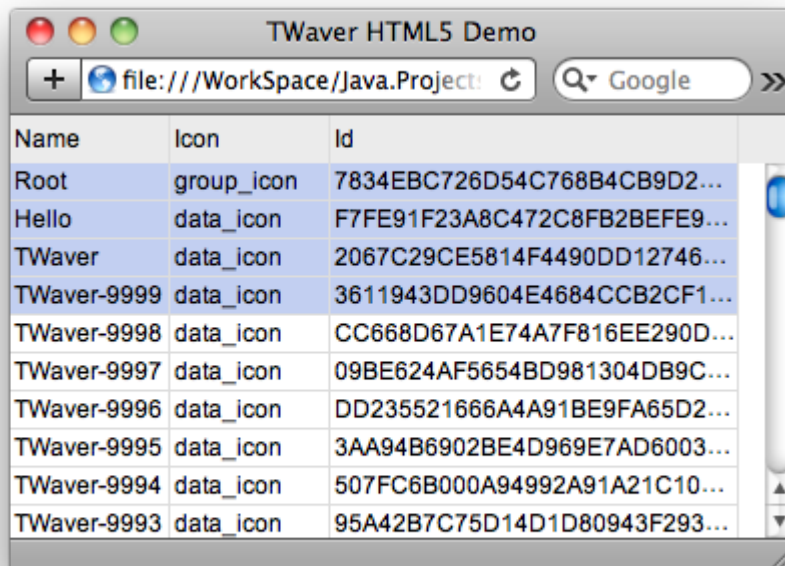
运行结果



Table组件介绍

twaver.controls.Table组件也称为表格组件，实现了与DataBox数据容器的绑定，显示其中的数据元素，每一行对应一个元素，每一列对应元素的一个属性。

table示例截图



Name	Icon	Id
Root	group_icon	7834EBC726D54C768B4CB9D2...
Hello	data_icon	F7FE91F23A8C472C8FB2BEFE9...
TWaver	data_icon	2067C29CE5814F4490DD12746...
TWaver-9999	data_icon	3611943DD9604E4684CCB2CF1...
TWaver-9998	data_icon	CC668D67A1E74A7F816EE290D...
TWaver-9997	data_icon	09BE624AF5654BD981304DB9C...
TWaver-9996	data_icon	DD235521666A4A91BE9FA65D2...
TWaver-9995	data_icon	3AA94B6902BE4D969E7AD6003...
TWaver-9994	data_icon	507FC6B000A94992A91A21C10...
TWaver-9993	data_icon	95A42B7C75D14D1D80943F293...

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var table = new twaver.controls.Table();
      var box = table.getDataBox();

      var i = 10000;
      while(i-->0){
        data = new twaver.Data();
        data.setName("TWaver-" + i);
        data.setParent(root);
        box.add(data);
      }

      var tablePane = new twaver.controls.TablePane(table);
      createColumn(table,'Name', 'name', 'accessor', 'string');
      createColumn(table,'Id', 'id', 'accessor', 'string');
      createColumn(table,'Icon', 'icon', 'accessor');

      var tableDom = tablePane.getView();
      tableDom.style.width = "100%";
      tableDom.style.height = "100%";
      document.body.appendChild(tableDom);
    }
  </script>
</head>
</html>
```

```
function createColumn(table, name, propertyName, propertyType, valueType) {  
    var column = new twaver.Column(name);  
    column.setName(name);  
    column.setPropertyName(propertyName);  
    column.setPropertyType(propertyType);  
    if (valueType) column.setValueType(valueType);  
    table.getColumnBox().add(column);  
    return column;  
}  
</script>  
</head>  
<body onload="init()" style="margin:0;" > </body>  
</html>
```

数据序列化

TWaver HTML5支持数据容器的导入导出，以便于数据的保存和传输，实现类为twaver.XMLSerializer，它提供了serialize/deserialize方法，用于数据的序列化和反序列化。

构建一个XMLSerializer

```
var xmlSerializer = new twaver.XMLSerializer(box);
```

序列化与反序列化

```
serialize : function()  
deserialize : function(xml, rootParent)
```

XMLSerializer还可以设置哪些属性可以输出，哪些不输出
不输出元素id

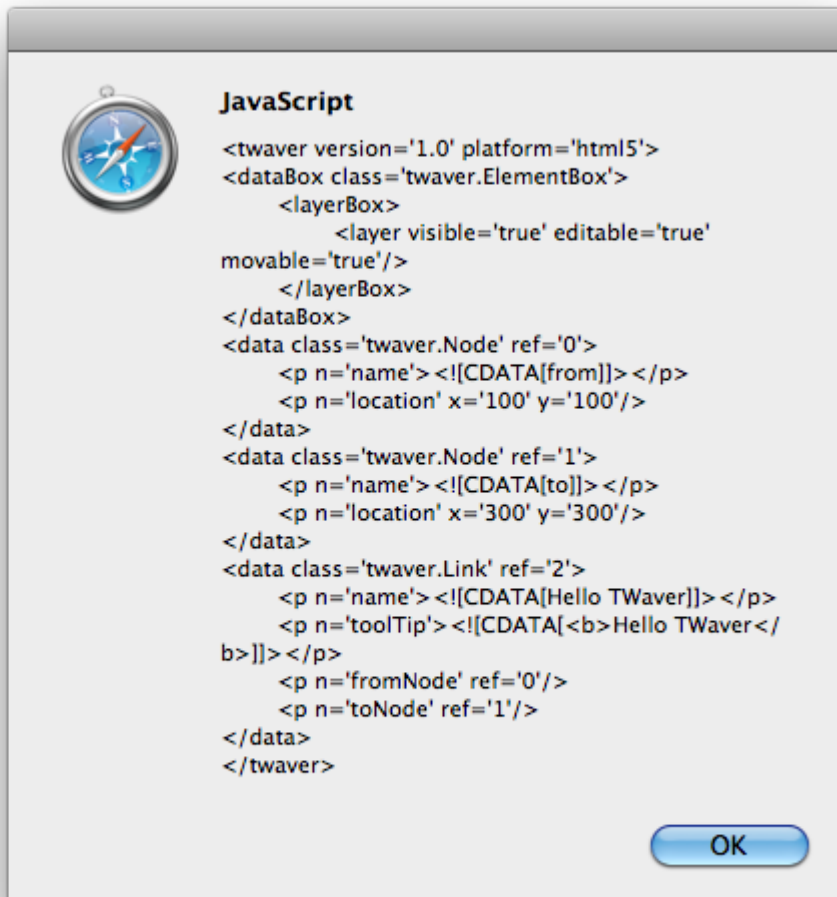
```
xmlSerializer.settings.registerProperty("id",null);
```

测试例子：

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>TWaver HTML5 Demo</title>  
  <script type="text/javascript" src="../demo/twaver.js"></script>  
  <script type="text/javascript">  
    function init(){  
      var network = new twaver.network.Network();  
  
      var box = network.getElementBox();  
      var node = new twaver.Node();  
      node.setName("from");  
      node.setLocation(100, 100);  
      box.add(node);  
      var node2 = new twaver.Node();  
      node2.setName("to");  
      node2.setLocation(300, 300);  
      box.add(node2);  
  
      var link = new twaver.Link(node, node2);  
      link.setName("Hello TWaver");  
      link.setToolTip("<b>Hello TWaver</b>");  
      box.add(link);  
  
      var xmlSerializer = new twaver.XMLSerializer(box);  
      xmlSerializer.settings.registerProperty("id",null);  
      alert(xmlSerializer.serialize());  
  
      var networkDom = network.getView();  
      networkDom.style.width = "100%";  
      networkDom.style.height = "100%";  
      document.body.appendChild(networkDom);  
    }  
  </script>  
</head>  
</html>
```

```
</script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

输出结果：



数据元素

数据元素是数据模型的基本要素，用于描述图形网元，业务网元，或者纯数据。TWaver HTML5中所有数据元素都继承自twaver.Data。为不同功能的需要，预定义了三类数据类型twaver.Element, twaver.Alarm, twaver.Layer，分别用来描述拓扑网元，告警和图层。其中拓扑网元扩展定义了十几种网元类型，用以描述丰富的拓扑网元特性，其中最常用的几类拓扑网元包括：基本网元，连线，子网，分组，设备面板...本章将详细介绍这些网元以及其他数据类型的特性，使用与扩展应用

- [基本数据元素](#)
- [告警元素](#)
- [图层元素](#)
- [拓扑元素](#)
 - [twaver.Node](#)
 - [twaver.Link](#)
 - [连线类型](#)
 - [连线绑定](#)
 - [twaver.Follower](#)
 - [twaver.Grid](#)
 - [twaver.Group](#)
 - [twaver.SubNetwork](#)
 - [twaver.ShapeNode](#)

基本数据元素

twaver.Data是TWaver HTML5中最基本的数据元素，默认定义了id, name, icon, toolTip, parent, children等基本属性，支持事件派发与监听，由它扩展出的还有twaver.Element，twaver.Alarm，twaver.Layer等数据类型。

事件派发与监听

Data包含一个twaver.EventDispatcher实例对象，这使它具有事件派发和监听的功能，可以通过调用下面的方法派发事件或者添加实现监听器：

```
firePropertyChange: function (property, oldValue, newValue)
addPropertyChangeListener : function (listener, scope, ahead)
removePropertyChangeListener : function (listener)
onPropertyChanged : function(listener)
```

基本属性

定义了一些基本属性，包括编号，名称，图标，提示文本.....

其中id是数据元素在数据容器中的唯一标识，不能重复，构造Data时，TWaver会自动设置唯一编号，当然用户也可以自己传入，id设置后不能再修改

```
twaver.Data : function (id)

getId : function()
getName : function()
setName : function(value)
getIcon : function()
setIcon : function(value)
getToolTip : function()
setToolTip : function(value)
```

如果用户需要设置其他属性，可以通过setClient(...)方法，类似Java中的HashMap存放元素属性

```
setClient = function (clientProp, newValue)
getClient = function (clientProp)
```

其他功能函数

此外，Data中还定义了其他功能函数

```
getChildren: function ()
getChildrenSize: function ()
toChildren: function (matchFunction, scope)
addChild: function (child, index)
removeChild: function (child)
getChildAt: function (index)
clearChildren: function ()
getParent: function ()
setParent: function (parent)
hasChildren: function ()
isRelatedTo: function (data)
```

isParentOf: [function](#) (data)
isDescendantOf: [function](#) (data)

告警元素

TWaver中定义了告警，每个告警有告警级别，用以反映告警的紧急程度，告警使用AlarmBox进行管理，将告警与拓扑网元相关联。网元本身不存储具体的告警，而只存储当前告警状态信息。

告警 - twaver.Alarm

用来表示网管系统中设备故障或者网络异常的数据模型，与Element关联以反映网元的告警信息，Alarm中预定义了告警级别、告警是否已清除，告警是否已确认以及发出告警的网元编号，用户也可以通过setClient(...)方法添加自己的属性。

告警中定义的属性如下：

```
twaver.Alarm : function (alarmID, elementID, isAcked, isCleared)
getElementID : function()
isAcked : function()
setAcked : function(value)
isCleared : function()
setCleared : function(value)
getAlarmSeverity : function()
setAlarmSeverity : function(value)
```

告警级别 - twaver.AlarmSeverity

告警级别用以反映告警的紧急程度，TWaverHTML5中预定义了六种告警级别，告警级别的value属性表示告警的严重程度，默认值越大告警越严重

```
twaver.AlarmSeverity : function (value, name, nickName, color, displayName)
twaver.AlarmSeverity.CRITICAL = twaver.AlarmSeverity.add(500, "Critical", "C", '#FF0000');
twaver.AlarmSeverity.MAJOR = twaver.AlarmSeverity.add(400, "Major", "M", '#FFA000');
twaver.AlarmSeverity.MINOR = twaver.AlarmSeverity.add(300, "Minor", "m", '#FFFF00');
twaver.AlarmSeverity.WARNING = twaver.AlarmSeverity.add(200, "Warning", "W", '#00FFFF');
twaver.AlarmSeverity.INDETERMINATE = twaver.AlarmSeverity.add(100, "Indeterminate", "N", '#C800FF');
twaver.AlarmSeverity.CLEARED = twaver.AlarmSeverity.add(0, "Cleared", "R", '#00FF00');
```

Severity	Letter	Value	Color
CRITICAL	C	500	Red
MAJOR	M	400	Orange
MINOR	m	300	Yellow
WARNING	W	200	Cyan
INDETERMINATE	N	100	Purple
CLEARED	R	0	Green

告警级别中的级别都是静态变量，用户也可以全局注册或者卸载自己的告警级别，此外还提供清除所有告警级别的方法（因为是全局变量，删除告警级别会对整个程序影响，要小心使用）

```
twaver.AlarmSeverity.add : function (value, name, nickName, color, displayName)
twaver.AlarmSeverity.remove : function (name)
```

```
twaver.AlarmSeverity.clear : function ()
```

告警状态AlarmState

实际应用中，告警的出现可能成千上万，面对告警风暴，直接与告警关联是沉重的，所以TWaver采用网元与告警分离，由告警容器去管理所有告警，网元本身只存储告警状态信息，如当前有多少条告警，最高级别是什么，而对于每条告警具体的信息存放在告警容器(AlarmBox)中。

网元告警状态用 twaver.AlarmState来定义，用来反映新发告警的级别和数量。

告警状态属性包括，确认告警的最高级别，新发告警的最高级别，该网元所有告警的最高级别，新发告警次高级别，自身告警最高级别，传递告警级别以及各级别告警的数量

```
getHighestAcknowledgedAlarmSeverity: function ()
getHighestNewAlarmSeverity: function ()
getHighestOverallAlarmSeverity: function ()
getHighestNativeAlarmSeverity: function ()
hasLessSevereNewAlarms: function ()
getAcknowledgedAlarmCount: function (severity)
getAlarmCount: function (severity)
getNewAlarmCount: function (severity)
setNewAlarmCount: function (severity, count)
getPropagateSeverity: function ()
setPropagateSeverity: function (propagateSeverity)
isEmpty: function ()
isEnabledPropagation: function ()
setEnabledPropagation: function (enablePropagation)
```

修改告警状态的相关方法：确认告警，清除告警，增加/减少确认告警，删除告警...

```
acknowledgeAlarm: function (severity)
acknowledgeAllAlarms: function (severity)
increaseAcknowledgedAlarm: function (severity, increment)
increaseNewAlarm: function (severity, increment)
decreaseAcknowledgedAlarm: function (severity, decrement)
decreaseNewAlarm: function (severity, decrement)
removeAllNewAlarms: function (severity)
setAcknowledgedAlarmCount: function (severity, count)
removeAllAcknowledgedAlarms: function (severity)
clear: function ()
```

告警的使用

在使用告警时需要注意一点，告警增删都要通过alarmBox来操作，这点与网元需要在elementBox中增删是一致的

示例：

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo - Alarm</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var network = new twaver.network.Network();

      var box = network.getElementBox();
```

```

var node = new twaver.Node();
node.setName("from");
node.setLocation(100, 100);
box.add(node);

addAlarm("alarm 1",node.getId(),twaver.AlarmSeverity.CRITICAL, box.getAlarmBox());

var networkDom = network.getView();
networkDom.style.width = "100%";
networkDom.style.height = "100%";
document.body.appendChild(networkDom);
}
function addAlarm(alarmID,elementID,alarmSeverity,alarmBox){
    var alarm = new twaver.Alarm(alarmID,elementID,alarmSeverity);
    alarmBox.add(alarm);
}
</script>
</head>
<body onload="init()" style="margin:0;" > </body>
</html>

```



图层元素

twaver.Layer (图层) , 用于描述拓扑网元的图层信息 , Layer有三个特殊属性 : visible, editable, movable.

```
getVisible : function()
setVisible : function(value)
getMovable : function()
setMovable : function(value)
getEditable : function()
setEditable : function(value)
```

TWaverHTML5中的层次关系由LayerBox来管理, 默认的层次顺序由父子关系和加入的先后顺序决定。拓扑图中, 每个Element通过设置layerId与某个layer相关联以控制网元的显示层次。

下面的例子展示了图层的使用以及图层三个属性的作用: 图层的上下移动等更多说明请参考LayerBox

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo - Layer</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var layerBox, box;
    function init() {
      var network = new twaver.network.Network();

      box = network.getElementBox();
      layerBox = box.getLayerBox();

      var layer1 = new twaver.Layer("unmovable", "unmovable layer");
      layer1.setMovable(false);
      var layer2 = new twaver.Layer("uneditable", "uneditable layer");
      layer2.setEditable(false);
      var layer3 = new twaver.Layer("invisible", "invisible layer");
      layer3.setVisible(false);

      layerBox.add(layer1);
      layerBox.add(layer2);
      layerBox.add(layer3, 0);

      createNode(layer1, "circle", 10, 40, 100, 100, "#ff0000");
      createNode(layer2, "diamond", 30, 60, 100, 100, "#00ff00");
      createNode(layer3, "rectangle", 50, 80, 100, 100, "#0000ff");
      createNode(layerBox.getDefaultLayer(), "rectangle", 70, 20, 150, 150, "#808080");

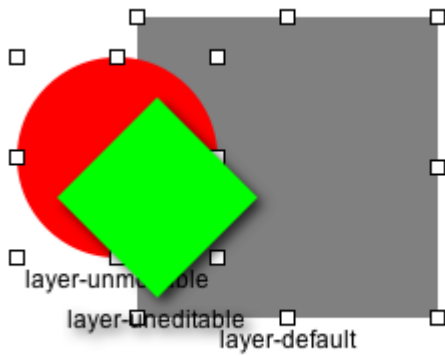
      var networkDom = network.getView();
      networkDom.style.width = "100%";
      networkDom.style.height = "100%";
      document.body.appendChild(networkDom);

      network.setEditInteractions();
    }
    function createNode(layer, shape, x, y, width, height, fillColor) {
      var node = new twaver.Node();
      node.setLayerId(layer.getId());
      node.setName("layer-" + layer.getId());
      node.setStyle("body.type", "vector");
      node.setStyle("vector.fill.alpha", 0.7);
      node.setStyle("vector.shape", shape);
      node.setSize(width, height);
    }
  </script>
</head>
</html>
```

```

node.setLocation(x, y);
node.setStyle("vector.fill.color", fillColor);
box.add(node);
return node;
}
</script>
</head>
<body onload="init()" style="margin:0;" > </body>
</html>

```



拓扑元素

TWaverHTML5中的twaver.Element表示拓扑网元，是TWaver中最重要的数据元素类型。拓扑元素用于拓扑图，主要分三大类，哑节点，节点，连线。

哑节点

其中哑节点在拓扑图上不可见，在树组件上可见，通常设置为其他节点的父节点，表示类别或分组，如将所有的Link类型网元放在一个Dummy节点下，表示Link分类。

节点

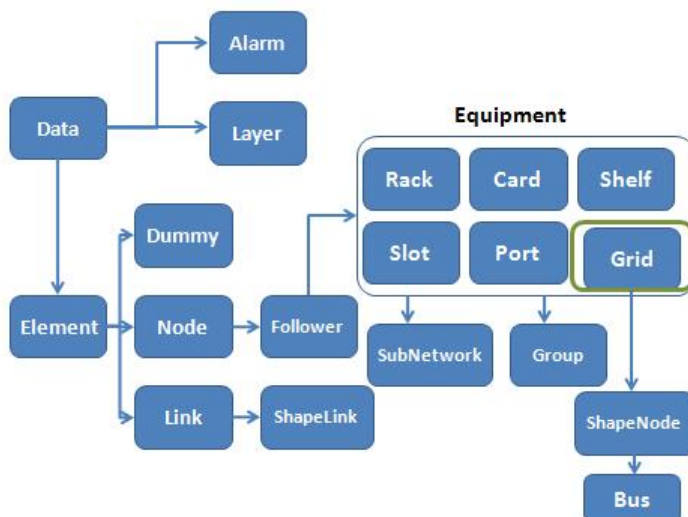
节点是最常用的网元类型，表示实体对象，包括节点，分组，子网，设备.....

连线

连线表示节点之间的连接关系，ShapeLink继承于Link，可以用于表示不规则走向的连线。

数据元素结构图

下图中Element部分为拓扑元素的继承结构，其中设备面板元素类型在TWaverHTML5 demo中提供



Element

Element继承于Data接口，扩展定义了alarmState, layerID, elementUIClass等属性，分别表示网元的告警状态，图层编号，视图类型

```
getLayerID : function()
setLayerID : function(layerId)
getAlarmState: function ()
getElementUIClass: function ()
```

其中的elementUIClass为ElementUI类型及其扩展类型，不同类型网元通常对应不同类型的ElementUI类，如twaver.Node对应于twaver.network.NodeUI，twaver.Grid对应twaver.network.GridUI。ElementUI是网元在拓扑图中的视图组件，两者构成一个数据与视图分离的模型结构，关于ElementUI将在后面的视图章节详细介绍。Element继承了IStyle接口，定义了get/setStyle()方法用于设置网元样式，包括颜色，边框，背景，对齐方式等，通常不同的网元有不同的样式属性，样式列表请参考附录

```
setStyle : function(styleProp, newValue)
getStyle : function(styleProp, returnDefaultIfNull)
```

下面的网元设置标签颜色为红色：

```
var node = new twaver.Node();
node.setStyle("label.color", "#ff0000");
node.setName("Node A");
```

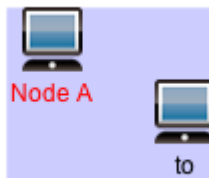
效果如下：



此外IElement还定义了isAdjustedToBottom()方法，表示是否置底显示，默认为false

```
isAdjustedToBottom: function ()
```

twaver.Group类型网元重写了这个方法，返回true，表示置底显示，这样就不会覆盖其孩子节点，而是位于其孩子节点之下显示，如下效果：



- [twaver.Node](#)
- [twaver.Link](#)
- [twaver.Follower](#)
- [twaver.Grid](#)
- [twaver.Group](#)
- [twaver.SubNetwork](#)
- [twaver.ShapeNode](#)

twaver.Node

twaver.Node继承于twaver.Element，表示拓扑图上的节点。Node通常表示实体对象，可以作为连线的端点，通过x, y, width, height属性确定位置和尺寸，能设置图片等属性

图片注册

注意TWaver HTML5中所有的图片都需要先注册，如下：

```
twaver.Util.registerImage : function (name, source, width, height)
```

图片注册示例：

```
twaver.Util.registerImage('loading', 'images/loading.gif', 32, 32);  
var node3 = new twaver.Node();  
node3.setImage('loading');
```

获取图片尺寸

需要注意的是，图片注册时需要指定宽高，比如32X32，当然也有图片大小预先不可知的情况，这时候可以采用下面的代码处理，先创建一个Image对象，监听Image加载事件，在图片被加载后，可以获取其宽高，这时再注册图片，如下示例：

```
registerImage: function (url) {  
    var image = new Image();  
    image.src = url;  
    image.onload = function () {  
        twaver.Util.registerImage(demo.Util.getImageName(url), image, image.width, image.height);  
        image.onload = null;  
        if (window.network) {  
            window.network.invalidateElementUIs();  
        }  
        if (window.tree) {  
            window.tree.invalidateDisplay();  
        }  
    };  
}
```

示例

下面我们看一个图片注册和使用的完整示例

```
<!DOCTYPE html>  
<html>  
<head>  
    <title>TWaver HTML5 Demo</title>  
    <script type="text/javascript" src="../demo/twaver.js"></script>  
    <script type="text/javascript">  
        function init() {  
            registerImage("images/twaver.png", "twaver");  
  
            var network = new twaver.network.Network();
```

```
var box = network.getElementBox();
var node = new twaver.Node();
node.setImage("twaver");
node.setName("from");
node.setLocation(100, 100);
box.add(node);

var node2 = new twaver.Node();
node2.setName("to");
node2.setLocation(200, 200);
box.add(node2);

var networkDom = network.getView();
networkDom.style.width = "100%";
networkDom.style.height = "100%";
document.body.appendChild(networkDom);
}
function registerImage(url, name) {
    var image = new Image();
    image.src = url;
    image.onload = function() {
        twaver.Util.registerImage(name, image, image.width, image.height);
        image.onload = null;
        if (window.network) {
            window.network.invalidateElementUIs();
        }
        if (window.tree) {
            window.tree.invalidateDisplay();
        }
    };
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>
```

运行界面


from


to

节点其他属性和方法

节点的位置和尺寸相关属性和方法

```
get/setX : function(x)
get/setY : function(y)
get/setCenterLocation : function(point)
get/setLocation : function(point)
```

```
get/setWidth : function(width)
get/setHeight : function(height)
translate: function (x, y)
```

获取与节点相连的连线，如果没有连线与节点相连，返回的是null

```
getLoopedLinks: function ()
getLinks: function ()
getAgentLinks: function ()
getFromLinks: function ()
getToLinks: function ()
hasAgentLinks: function ()
getFromAgentLinks: function ()
getToAgentLinks: function ()
```

此外节点上还可以添加跟随者，跟随者能跟随节点的移动而移动，我们将在Follower章节详细介绍

```
getFollowers: function ()
```

twaver.Link

连线通常表示为节点间的连接关系，需要指定起始结束节点，TWaver支持自环，即起始结束节点为同一个节点。

起始节点，结束节点和连线

```
get/setFromNode: function (fromNode)
get/setToNode: function (toNode)
```

起始代理节点和结束代理节点

直接与连线相连的节点我们称为fromNode，toNode，如果起始或者结束节点放在分组中，分组合并状态时，外观上分组与连线相连，这时此分组为代理节点

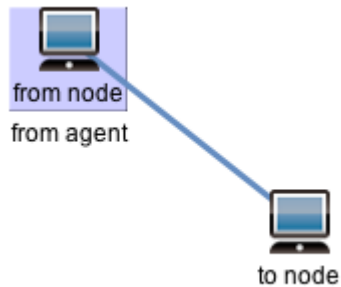
```
getFromAgent: function ()
getToAgent: function ()
```

示例：

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
function init() {
  var network = new twaver.network.Network();
  var box = network.getElementBox();

  var node1=new twaver.Node();
  node1.setLocation(20,20);
  node1.setName("from node");
  var node2=new twaver.Node();
  node2.setLocation(150,50);
  node2.setName("to node");
  var group=new twaver.Group();
  group.addChild(node1);
  group.setName("from agent");
  var link6=new twaver.Link(node1,node2);
  box.add(node1);
  box.add(node2);
  box.add(group);
  box.add(link6);

  var networkDom = network.getView();
  networkDom.style.width = "100%";
  networkDom.style.height = "100%";
  document.body.appendChild(networkDom);
}
</script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```



自环，起始结束节点为同一个节点时我们称为自环

```
isLooped: function ()
```

示例：

```
var node = new twaver.Node();
box.add(node);
var link = new twaver.Link(node, node);
link.setName("Looped Link");
box.add(link);
```



连线的展开和捆绑

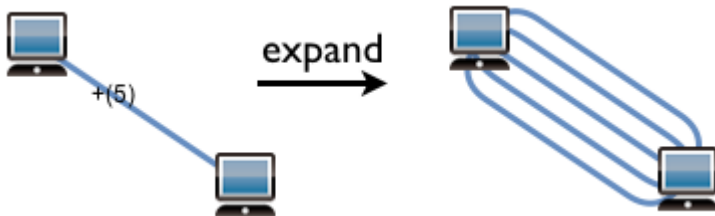
当节点之间有多条连线时，TWaver支持连线的捆绑和展开，默认双击其中一条连线即可实现两种状态的切换，绑定时显示的连线称为绑定代理，默认将第一条连线设为代理，用户也可以指定代理

```
getBundleLinks: function ()
getBundleCount: function ()
getBundleIndex: function ()
reverseBundleExpanded: function ()
isBundleAgent: function ()
```

示例：

```
var from = new twaver.Node();
from.name = "from";
from.location = {x:20, y:20};
box.add(from);
var to = new twaver.Node();
to.name = "to";
to.location = {x:150, y:60};
box.add(to);
var link = new twaver.Link(from,to);
link.name = "bundle agent";
```

```
box.add(link);
var link2=new twaver.Link(from,to);
box.add(link2);
var link3=new twaver.Link(from,to);
box.add(link3);
var link4=new twaver.Link(from,to);
box.add(link4);
var link5=new twaver.Link(from,to);
box.add(link5);
```



- [连线类型](#)
- [连线绑定](#)

连线类型

连线用于连接两个节点，可表示为链路，管线等业务对象，TWaver HTML5中的连线类为twaver.Link，由之扩展出来的有twaver.ShapeLink，这种连线是由一系列控制点决定连线走向。如果起始和结束节点为同一节点，也就是节点A连向节点A，这种情况我们成为自环。

twaver.Link

普通连线，也就是twaver.Link，它提供了几种走向类型，主要为三类：

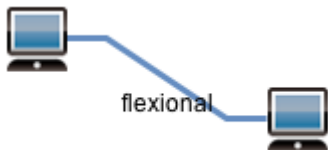
直线

直接连接起始结束节点
'arc', 'triangle', 'parallel'



延伸直线

先从端点的水平或者垂直方向延伸，然后直线连接
'flexional', 'flexional.horizontal', 'flexional.vertical'



正交直线

这种连线最为丰富，可以通过一个控制点走正交连线
'orthogonal', 'orthogonal.horizontal', 'orthogonal.vertical', 'orthogonal.H.V', 'orthogonal.V.H', 'extend.top', 'extend.left', 'extend.bottom', 'extend.right'



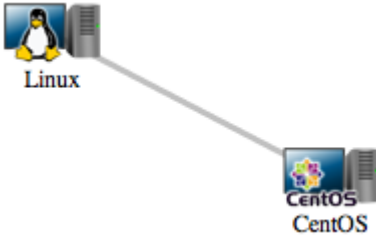

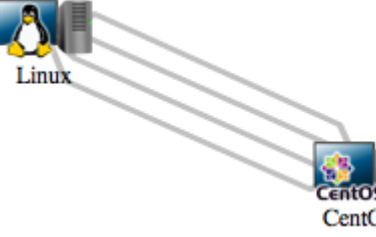

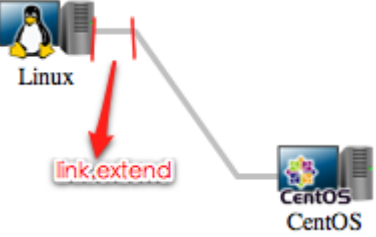
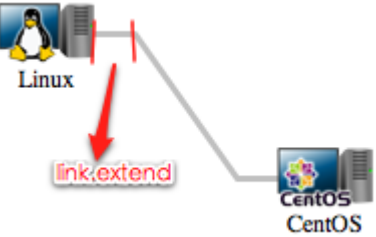
下面我们详细介绍这几种连线类型的使用和特点

设置连线类型

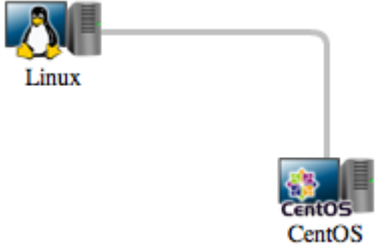
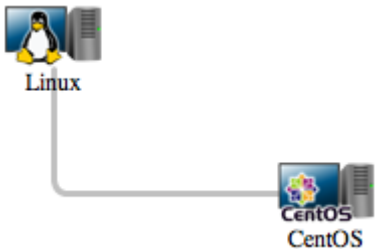
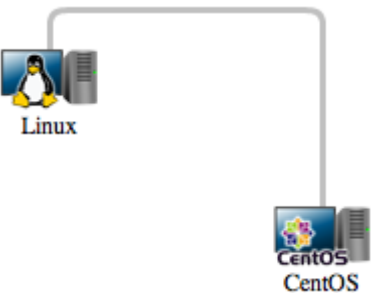
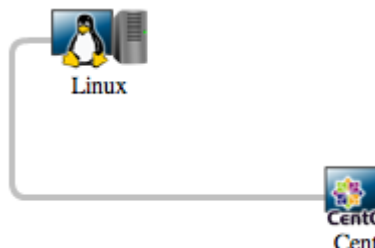

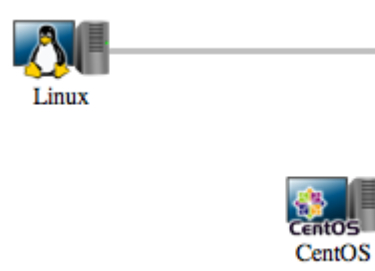
```
link.setStyle('link.type', 'orthogonal');
```

下面列表列举了各种连线类型以及其相应的控制参数

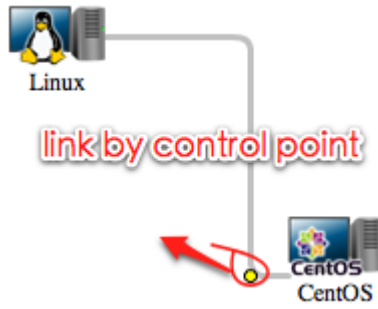
Link Type	呈现	控制参数
-----------	----	------

<p>基本类型 直接连接起始结束节点 两节点间存在多条连线时， 按连线展开效果又分为下面三种类型： arc, triangle, parallel 这些类型只有在两节点之间存在多条 连线时， 呈现有区别</p>		<p>link.from.position 默认center link.from.xoffset 默认0 link.from.yoffset 默认0 link.to.position 默认center link.to.xoffset 默认0 link.to.yoffset 默认0</p>
<p>arc 连线展开时，拐点呈圆弧状</p>		<p>link.bundle.offset 拐点偏移量，默认值20 link.bundle.gap 连线间距，默认值12</p>
<p>triangle 连线展开时，拐点处直线相连</p>		<p>同上</p>
<p>parallel 连线展开时，呈现平行线</p>		<p>同上</p>
<p>flexional 此类连线如右图所示，按方向分三种： 自动方向： 若两节点间X方向间隙大，则取水平 方向 水平方向： flexional.horizontal 垂直方向： flexional.vertical</p>		<p>link.from.at.edge 连接到起始节点的边缘 默认为true link.to.at.edge 连接到结束节点的边缘 默认值为true link.extend 展开距离，默认值20</p>
<p>flexional.horizontal</p>		<p>同上</p>

flexional.vertical		同上
正交直线连线类型		
<p>orthogonal</p> <p>正交直线</p> <p>按方向分三类：</p> <p>自动方向：</p> <p>若两节点间X方向间隙大，则取水平方向</p> <p>水平方向：</p> <p>orthogonal.horizontal</p> <p>垂直方向：</p> <p>orthogonal.vertical</p>		<p>link.from.at.edge 连接到起始节点的边缘 默认为true link.to.at.edge 连接到结束节点的边缘 默认为true link.split.by.percent 是否按百分比劈分，默认为true link.split.percent 劈分点距起始节点的百分比位置 默认值为0.5 若按百分比劈分，请设置此属性 link.split.value 劈分点距起始节点的偏移量 默认值为20 若按偏移量劈分，请设置此属性</p>
orthogonal.horizontal		同上
orthogonal.vertical		同上

<p>orthogonal.vertical 从起始节点开始，先沿水平方向，后垂直方向</p>		<p>link.from.at.edge 连接到起始节点的边缘 默认为true link.to.at.edge 连接到结束节点的边缘 默认值为true</p>
<p>vertical.horizontal 从起始节点开始，先沿垂直方向，后水平方向</p>		
<p>extend.top 向上扩展 扩展量为劈分点到连线起始结束最顶端的距离</p>		<p>link.extend 扩展量，默认为20</p>
<p>extend.left 向左扩展 扩展量为劈分点到连线起始结束最左端的距离</p>		<p>同上</p>
<p>extend.bottom 向底扩展 扩展量为劈分点到连线起始结束最底端的距离</p>		<p>同上</p>
<p>extend.right 向右扩展 扩展量为劈分点到连线起始结束最右端的距离</p>		<p>同上</p>

上面的正交直线类型都可以按控制点
确定走向
如果连线设置了控制点，
优先使用控制点来决定劈分点位置



link.control.point
控制点
连线劈分点位置由此控制点决定
连线方向由连线类型决定
自动方向：
orthogonal
水平方向：
orthogonal.horizontal
垂直方向：
orthogonal.vertical

连线绑定

twaver.Link章节我们简要介绍了连线的绑定与展开，本章将详细介绍分组绑定，自环绑定，绑定与展开以及展开间隙等属性

普通连线绑定

TWaver定义了下面几种连线绑定相关参数：

```
"link.bundle.id" : 绑定分组编号，同一编号的连线组成一组  
"link.bundle.independent" : 分组是否独立，存在多个连线分组时，是否独立绑定  
"link.bundle.gap" : 连线之间的间隙  
"link.bundle.offset" : 连线离端点的偏移量  
"link.bundle.enable" : 是否参与绑定  
"link.bundle.expanded" : 是否为展开状态，false表示状态为绑定
```

分组绑定

下面的例子中，定义了两组连线，分别设置bundlerID为"1"和"2"

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>TWaver HTML5 Demo </title>  
  <script type="text/javascript" src="../demo/twaver.js"></script>  
  <script type="text/javascript">  
var box;  
function init() {  
  var network = new twaver.network.Network();  
  box = network.getElementBox();  
  
  var from = new twaver.Node();  
  from.setName("from");  
  from.setLocation({  
    x : 20,  
    y : 20  
  });  
  box.add(from);  
  var to = new twaver.Node();  
  to.setName("to");  
  to.setLocation({  
    x : 150,  
    y : 60  
  });  
  box.add(to);  
  
  createLink(from, to, "g1_1", 1);  
  createLink(from, to, "g1_2", 1);  
  createLink(from, to, "g1_3", 1);  
  
  createLink(from, to, "g2_1", 2, "#00ff00");  
  createLink(from, to, "g2_2", 2, "#00ff00");  
  createLink(from, to, "g2_3", 2, "#00ff00");  
  
  var networkDom = network.getView();  
  networkDom.style.width = "100%";  
  networkDom.style.height = "100%";  
  document.body.appendChild(networkDom);  
}
```

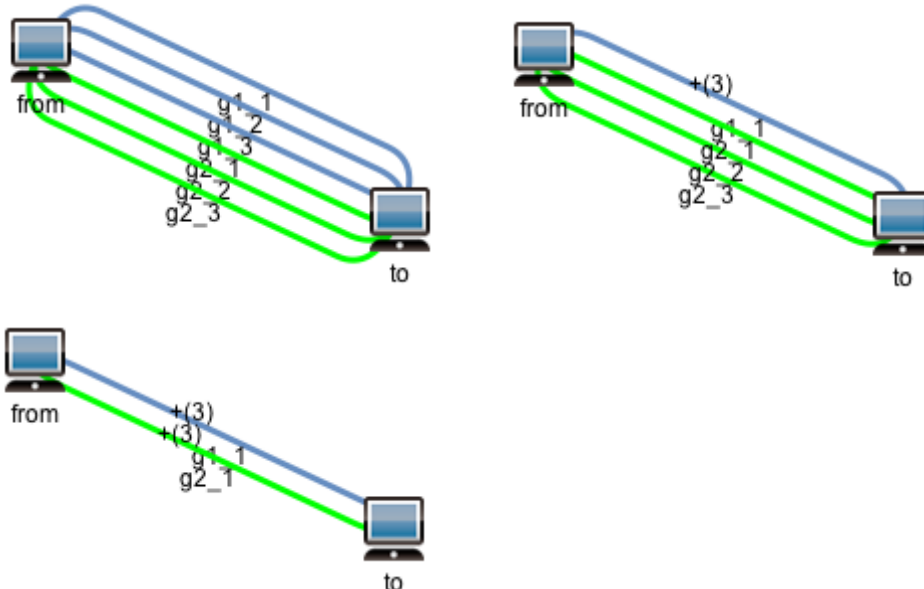
```

}

function createLink(from, to, name, groupID, color, type, groupIndependent,
    gap, offset, bundleEnable) {
    var link = new twaver.Link(from, to);
    link.setName(name);
    if (type) {
        link.setStyle("link.type", type);
    }
    if (color) {
        link.setStyle("link.color", color);
    }
    if (groupID >= 0) {
        link.setStyle("link.bundle.id", groupID);
    }
    if (gap > 0) {
        link.setStyle("link.bundle.gap", gap);
    }
    if (offset > 0) {
        link.setStyle("link.bundle.offset", offset);
    }
    link.setStyle("link.bundle.independent", groupIndependent);
    link.setStyle("link.bundle.enable", bundleEnable);
    box.add(link);
    return link;
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```

显示如下：



独立分组绑定

下面我们设置两组连线各自独立绑定，并分别设置不同的连线类型

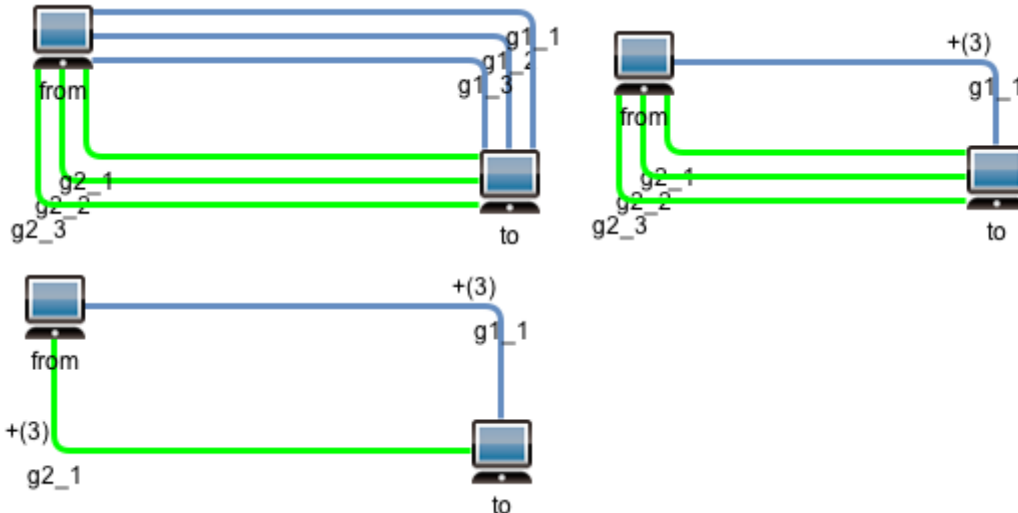
```

createLink(from, to, "g1_1", 1, null, "orthogonal.H.V");
createLink(from, to, "g1_2", 1, null, "orthogonal.H.V");
createLink(from, to, "g1_3", 1, null, "orthogonal.H.V");

```

```
createLink(from, to, "g2_1", 2, "#00ff00", "orthogonal.V.H", true);
createLink(from, to, "g2_2", 2, "#00ff00", "orthogonal.V.H", true);
createLink(from, to, "g2_3", 2, "#00ff00", "orthogonal.V.H", true);
```

呈现如下：



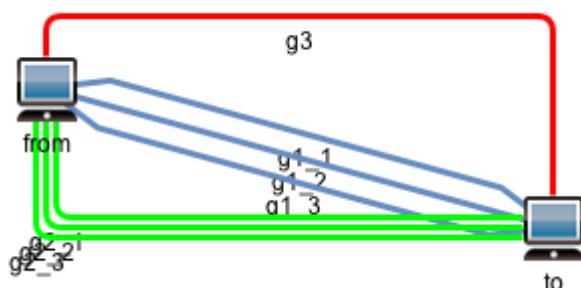
下面我们试一试"link.bundle.gap"和"link.bundle.offset"参数。另外通过"link.bundle.enable"参数，增加一条不参与绑定的连线

```
createLink(from, to, "g1_1", 1, null, "triangle", null, null, 30);
createLink(from, to, "g1_2", 1, null, "triangle", null, null, 30);
createLink(from, to, "g1_3", 1, null, "triangle", null, null, 30);

createLink(from, to, "g2_1", 2, "#00ff00", "orthogonal.V.H", true, 5);
createLink(from, to, "g2_2", 2, "#00ff00", "orthogonal.V.H", true, 5);
createLink(from, to, "g2_3", 2, "#00ff00", "orthogonal.V.H", true, 5);

createLink(from, to, "g3", 2, "#ff0000", "extend.top", null, null, null, false);
```

呈现如下：



在上面的例子基础上，我们使用api设置连线绑定状态为合并

```
createLink(from, to, "g1_1", 1, null, "triangle", null, null, 30);
createLink(from, to, "g1_2", 1, null, "triangle", null, null, 30);
createLink(from, to, "g1_3", 1, null, "triangle", null, null, 30);

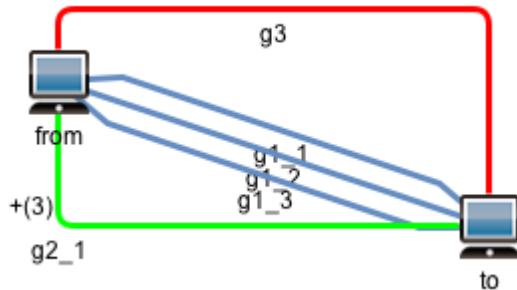
var link = createLink(from, to, "g2_1", 2, "#00ff00", "orthogonal.V.H", true, 5);
```



```
link.setStyle("link.bundle.expanded",false); // 设置连线为绑定状态
createLink(from, to, "g2_2", 2, "#00ff00", "orthogonal.V.H", true, 5);
createLink(from, to, "g2_3", 2, "#00ff00", "orthogonal.V.H", true, 5);

createLink(from, to, "g3", 2, "#ff0000", "extend.top", null, null, null, false);
```

呈现如下：



自环绑定

TWaver HTML5支持自环连线，表示网元内部连接关系，自环的绑定使用独立的一组参数控制

```
link.looped.gap" : 自环间距，表示环与环之间的间距，默认值为12
link.looped.direction : 子网方向，包括东南西北八个方向: 'northwest', 'north', 'northeast', 'east', 'west', 'south',
'southwest', 'southeast'
link.looped.type : 自环类型，支持圆弧和矩形: 'arc', 'rectangle'
```

下面的例子将创建三条自环连线，并设置间距为20，自环类型为rectangle

```
function init() {
    var network = new twaver.network.Network();
    var box = network.getElementBox();

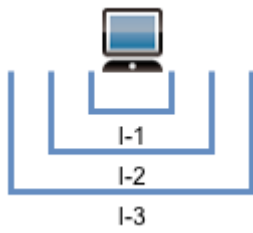
    var from = new twaver.Node();
    from.setLocation({
        x: 20,
        y: 20
    });
    box.add(from);

    var i = 3;
    while(i > 0){
        var link = new twaver.Link(from, from);
        link.setName("I-" + i);
        link.setStyle("link.looped.gap",20);
        link.setStyle("link.looped.direction", "south");
        link.setStyle("link.looped.type", "rectangle");
        box.add(link);
        i--;
    }

    var networkDom = network.getView();
    networkDom.style.width = "100%";
    networkDom.style.height = "100%";
    document.body.appendChild(networkDom);
```

```
}
```

呈现如下：



twaver.Follower

跟随者跟随宿主移动而移动，具有依附的功能，通常用于设备面板，如端口依附在板卡上

设置宿主节点

一个Node可以依附多个跟随者

```
//设置宿主  
get/setHost : function(host)  
//是否依附在某个节点上  
isHostOn : function(node)
```

互为宿主的情况

此外跟随者之间可以相互依附，即互为宿主和跟随者

```
//是否是相互依附  
isLoopedHostOn : function(node)
```

示例：

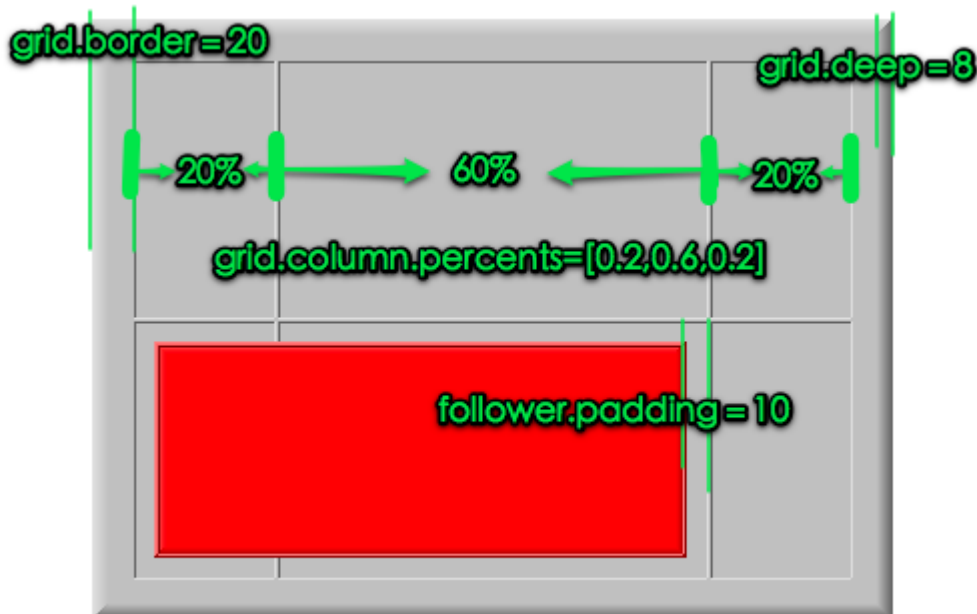
```
<!DOCTYPE html>  
<html>  
<head>  
  <title>TWaver HTML5 Demo</title>  
  <script type="text/javascript" src="../demo/twaver.js"></script>  
  <script type="text/javascript">  
function init() {  
  var network = new twaver.network.Network();  
  var box = network.getElementBox();  
  
  var node=new twaver.Node();  
  node.setLocation(20,20);  
  node.setName("Host");  
  box.add(node);  
  
  var follower = new twaver.Follower();  
  follower.setLocation(50,50);  
  follower.setHost(node);  
  follower.setName("Follower");  
  box.add(follower);  
  
  var networkDom = network.getView();  
  networkDom.style.width = "100%";  
  networkDom.style.height = "100%";  
  document.body.appendChild(networkDom);  
}  
  </script>  
</head>  
<body onload="init()" style="margin:0;"></body>  
</html>
```



twaver.Grid

twaver.Grid网元在拓扑图中表现为网格，实现了类似java Swing的TableLayout，通过指定行列号以及行列跨度实现网元的定位布局。

下面的示例是一个2行3列的网格，其中第二行一列单元格添加了一个孩子网元，此网元跨两列，并渲染为红色



示例：

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
function init() {
  var network = new twaver.network.Network();
  var box = network.getElementBox();

  var grid = new twaver.Grid();
  grid.setLocation(20, 20);
  grid.setSize(400, 300);
  grid.setStyle("grid.border", 20);
  grid.setStyle("grid.row.count", 2);
  grid.setStyle("grid.column.count", 3);
  grid.setStyle("grid.column.percents", [0.2, 0.6, 0.2]);
  grid.setStyle("grid.deep", 8);
  box.add(grid);

  var cell = new twaver.Grid();
  cell.setStyle("follower.column.index", 0);
  cell.setStyle("follower.row.index", 1);
  cell.setStyle("follower.column.span", 2);
  cell.setStyle("grid.fill.color", "#ff0000");
  cell.setStyle("follower.padding", 10);
  cell.setStyle("grid.deep", 5);
  cell.setHost(grid);
  grid.addChild(cell);
  box.add(cell);
}
```

```
var networkDom = network.getView();
networkDom.style.width = "100%";
networkDom.style.height = "100%";
document.body.appendChild(networkDom);
}
</script>
</head>
<body onload="init()" style="margin:0;" > </body>
</html>
```

twaver.Group

通常分组中包含多个孩子网元，分组展开时，同时显示分组和其下的孩子网元，合并时显示分组网元图标。

下面是分组嵌套以及展开合并的效果图
默认双击展开或合并分组



示例：

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
function init() {
  var network = new twaver.network.Network();
  var box = network.getElementBox();

  var node=new twaver.Node();
  node.setLocation(20,20);
  var node2=new twaver.Node();
  node2.setLocation(150,50);
  var link=new twaver.Link(node,node2);

  var childGroup=new twaver.Group();
  childGroup.addChild(node);
  childGroup.addChild(node2);
  childGroup.addChild(link);
  childGroup.setStyle("group.fill.color", "#ff0000");
  childGroup.setName("Child Group");
  childGroup.setExpanded(true);

  var group=new twaver.Group();
  group.addChild(childGroup);
  group.setStyle("group.fill.color", "#00ff00");
  group.setName("Group");
  group.setExpanded(true);

  box.add(node);
  box.add(node2);
  box.add(link);
  box.add(childGroup);
  box.add(group);

  var networkDom = network.getView();
  networkDom.style.width = "100%";
  networkDom.style.height = "100%";
  document.body.appendChild(networkDom);
}
</script>
</head>
<body onload="init()" style="margin:0;"></body>
```

```
</html>
```


twaver.SubNetwork

子网接口，表示大型网络中的一部分网络，TWaver HTML5中拓扑图可以切换子网，呈现的即是当前子网中的网元。

子网接口的默认实现类是twaver.SubNetwork，拓扑图中默认双击进入子网，双击背景返回上层子网，当前子网为Null时表示是最顶层子网。

Network中切换子网的方法：
twaver.network.Network

```
//设置当前子网  
get/setCurrentSubNetwork : function(subnetwork, animate)  
//返回上级子网  
upSubNetwork : function(animate)
```

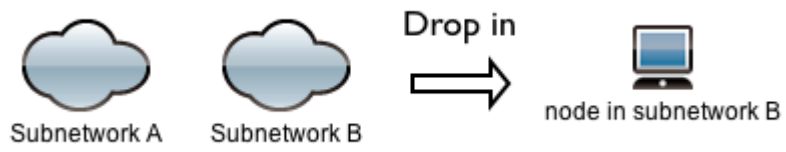
示例：

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>TWaver HTML5 Demo</title>  
  <script type="text/javascript" src="../demo/twaver.js"></script>  
  <script type="text/javascript">  
    function init() {  
      var network = new twaver.network.Network();  
      var box = network.getElementBox();  
  
      var node=new twaver.Node();  
      node.setLocation(20,20);  
      var node2=new twaver.Node();  
      node2.setLocation(150,50);  
      var link=new twaver.Link(node,node2);  
  
      var node=new twaver.Node();  
      node.setName("node in subnetwork A");  
      node.setLocation(100,50);  
      var subNetworkA=new twaver.SubNetwork();  
      subNetworkA.addChild(node);  
      subNetworkA.setName("Subnetwork A");  
      subNetworkA.setLocation(20,20);  
  
      var node2=new twaver.Node();  
      node2.setName("node in subnetwork B");  
      node2.setLocation(100,50);  
      var subNetworkB=new twaver.SubNetwork();  
      subNetworkB.addChild(node2);  
      subNetworkB.setName("Subnetwork B");  
      subNetworkB.setLocation(120,20);  
  
      box.add(node);  
      box.add(subNetworkA);  
      box.add(node2);  
      box.add(subNetworkB);  
  
      var networkDom = network.getView();  
      networkDom.style.width = "100%";  
      networkDom.style.height = "100%";  
      document.body.appendChild(networkDom);  
    }  
  </script>
```

```

</head>
<body onload="init()" style="margin:0;"></body>
</html>

```



twaver.ShapeNode

多边形节点由一系列控制点确定位置和形状，通常用于表现不规则图形，如覆盖范围，地图区块.....
添加删除控制点相关方法：

```
get/setPoint : function(points)
addPoint : function(point)
addPointAt : function(point, index)
setPointAt : function(point, index)
removePoint : function(point)
```

此外还可以指定片段的绘制方式（segments），使ShapeNode表现更丰富的效果，如曲线（QuadTo），间断（moveTo）等
绘制方式有三种，移动到（moveto），直线（lineto）和曲线（quadto）

```
"moveto", "lineto", "quadto"
```

设置segments
twaver.ShapeNode

```
get/setSegments : function(list)
```

使用如下：

首先添加五个控制点

然后设置片段的绘制方式，下面的例子中，首先移动到第一个控制点，然后画曲线（需要用两个控制点），然后再绘制曲线结束

```
var shapeNode2 = new twaver.ShapeNode();
shapeNode2.addPoint({
    x : 30,
    y : 10
});
shapeNode2.addPoint({
    x : 80,
    y : 10
});
shapeNode2.addPoint({
    x : 100,
    y : 90
});
shapeNode2.addPoint({
    x : 10,
    y : 90
});
shapeNode2.addPoint({
    x : 30,
    y : 10
});
shapeNode2.setStyle("vector.fill.color", "#00ff00");
shapeNode2.setLocation(150, 10);

var segments = new twaver.List();
segments.add("moveto");
```


```
segments.add("quadto");
segments.add("quadto");
shapeNode2.setSegments(segments);
shapeNode2.setName("ShapeNode with Segments");
box.add(shapeNode2);
```



ShapeNode



ShapeNode with Segments

 注意一个曲线片段需要占用两个控制点

示例：

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init() {
      var network = new twaver.network.Network();
      var box = network.getElementBox();

      var shapeNode=new twaver.ShapeNode();
      shapeNode.addPoint({x:30, y:10});
      shapeNode.addPoint({x:80, y:10});
      shapeNode.addPoint({x:100, y:90});
      shapeNode.addPoint({x:10, y:90});
      shapeNode.addPoint({x:30, y:10});
      shapeNode.setStyle("vector.fill.color", "#ff0000");
      shapeNode.setName("ShapeNode");
      box.add(shapeNode);

      var shapeNode2=new twaver.ShapeNode();
      shapeNode2.addPoint({x:30, y:10});
      shapeNode2.addPoint({x:80, y:10});
      shapeNode2.addPoint({x:100, y:90});
      shapeNode2.addPoint({x:10, y:90});
      shapeNode2.addPoint({x:30, y:10});
      shapeNode2.setStyle("vector.fill.color", "#00ff00");
      shapeNode2.setLocation(150, 10);

      var segments=new twaver.List();
      segments.add("moveto");
      segments.add("quadto");
      segments.add("quadto");
      shapeNode2.setSegments(segments);
      shapeNode2.setName("ShapeNode with Segments");
      box.add(shapeNode2);

      var networkDom = network.getView();
      networkDom.style.width = "100%";
      networkDom.style.height = "100%";
      document.body.appendChild(networkDom);
    }
  </script>
```

```
</head>  
<body onload="init()" style="margin:0;" ></body>  
</html>
```

数据容器

前面"基础"章节介绍了数据容器的种类和基本使用，本章将详细介绍数据容器使用过程中可能涉及到的概念和操作方式，包括数据容器中包含的各种事件类型，如何快速查找数据容器中的元素以及元素选中模式的使用和介绍

- [数据容器之事件机制](#)
- [快速查找 - QuickFinder](#)
- [选中模型 - SelectionModel](#)

数据容器之事件机制

TWaver的数据容器不仅包含元素集合，还管理着元素的层次关系，以及选中模型。当容器中的元素变化都会派发相应的变化事件，如元素增减变化派发DataBoxChangeEvent，元素属性变化派发PropertyChangeEvent，元素层次变化派发HierarchyChangeEvent，元素选中变化由SelectionModel管理，在"选中模型"章节将详细介绍

容器自身属性变化 - PropertyChange

databox自身可以设置属性，如设置databox名称，设置client属性，对于ElementBox还可以设置样式属性
设置数据容器自身属性

```
databox.setName("my databox");
databox.setClient(propertyName, value);
elementBox.setStyle(styleName, value);
...
```

数据容器的自身属性变化时，派发属性变化事件，通过下面的方式添加监听，捕获的事件对象包涵属性名称，新值和老值（property, oldValue, newValue）

```
//databox property change listener
databox.addPropertyChangeListener(function(e){ });
```

容器元素增减变化事件 - DataBoxChange

顾名思义，数据容器中元素数量变化时派发的事件，如添加新元素，删除元素，清除所有元素，分别对应添加事件，删除事件，清除事件，使用下面的方式添加监听容器变化事件监听

```
//databox change listener
databox.addDataBoxChangeListener(function(e){ });
```

捕获的事件对象包涵类型，数据等属性（kind, data, datas）

kind

容器变化事件分三种类型：add, remove, clear，分别表示增加数据，删除数据，清除数据

data

如果是添加或者删除事件，data属性表示当前被添加或删除的数据，如添加元素var dataA = new twaver.Data() 到dataBox，对应派发的添加事件的data属性就是dataA

datas

如果批量删除，目标对象是一个集合，可以通过datas属性获取

```
databox.addDataBoxChangeListener(function(e){
    var kind = e.kind;
    if(kind == "add"){
        console.log("add a new data [" + e.data + "]");
    }
});
```

元素属性变化事件 - DataPropertyChange

数据容器中元素的属性变化可以通过数据容器转发，这样就不用对每个元素添加属性变化监听，而是统一在数据容器上添加，捕获的事件对象包涵属性名称，新值和老值（property, oldValue, newValue）



注意元素属性变化事件与数据容器属性变化的区别，前者是元素属性变化，后者是databox自身属性变化

```
//data property change listener
databox.addDataPropertyChangeListener(function(e){ });
```

例如下面的代码可监听所有元素name变化事件

```
databox.addDataPropertyChangeListener(function(e){
    if(e.property == "name"){
        trace("data name changed - old name: " + e.oldValue + ", new name: " + e.newValue);
    }
});
```

层次变化事件 - HierarchyChange

根据元素的父子关系，数据容器中组织出层次逻辑，比如树视图的层次关系就是反应的databox的层次关系，我们可以通过修改元素的父子关系，或者调用databox中层次移动的相关函数实现元素层次的修改，当层次结构被修改时，数据容器就会派发层次变化事件

```
//hierarchy change listener
databox.addHierarchyChangeListener(function(e){ });
```

捕获的事件对象中包含两个重要信息，oldIndex, newIndex，分别代表原始次序位置 and 新的次序位置

实例分析：

下面的例子演示了databox上各种事件的监听和结果

```
var databox;
function init() {
    databox = new twaver.DataBox();

    // databox property change listener
    databox.addPropertyChangeListener(function(e) {
        console.log("databox property: " + e.property
            + " changed, old value: " + e.oldValue
            + ", new value: " + e.newValue + "");
    });

    // databox change listener
    databox.addDataBoxChangeListener(function(e) {
        if (e.data) {
            console.log("" + e.data.getName() + " " + e.kind);
        } else {
            console.log("databox " + e.kind);
        }
    });

    // data property change listener
    databox.addDataPropertyChangeListener(function(e) {
        console.log("data property: " + e.property
            + " changed, old value: " + e.oldValue
            + ", new value: " + e.newValue + "");
    });
}
```



```

    });

    // hiberarchy change listener
    databox.addHierarchyChangeListener(function(e) {
        console.log("data hierarchy changed, old index: '" + e.oldIndex
            + "', new index: '" + e.newIndex + "'");
    });

    databox.setName("Box A");

    var dataA = createData("A");
    var dataB = createData("B");
    dataA.setName("AA");
    dataB.setClient("NO", 1);
    databox.clear();

    var dataC = createData("C");
    var dataD = createData("D");
    var dataE = createData("E");
    var dataF = createData("F");

    databox.remove(dataC);

    databox.moveToTop(dataE);

    databox.moveToBottom(dataD);

    console.log("Root datas: " + databox.getRoots());
}

function createData(name) {
    var data = new twaver.Data();
    data.setName(name);
    databox.add(data);
    return data;
}

```

输出结果：

databox property: 'name' changed, old value: 'DataBox', new value: 'Box A'

'A' add

'B' add

data property: 'name' changed, old value: 'A', new value: 'AA'

data property: 'C:NO' changed, old value: 'undefined', new value: '1'

databox clear

'C' add

'D' add

'E' add

'F' add

'C' remove

data hierarchy changed, old index: '1', new index: '0'

data hierarchy changed, old index: '1', new index: '2'

Root datas: E,F,D

快速查找 - QuickFinder

快速查找用于快速查找指定属性值的所有元素，如查找某个告警级别的所有告警，特定类型的所有网元...

以查找指定名称元素为例

首先创建一个与name属性绑定的查找器：

```
var nameFinder = new twaver.QuickFinder(databox,"name");
```

构造函数第一个参数databox，是数据容器，查找器将在该数据容器中查找元素
第二个参数"name"，表示根据元素的name属性查找

使用方式：

```
var datas = nameFinder.find("group-1");
```

nameFinder.find("group-1") 表示查找所有name为"group-1"的元素，返回查找的元素集合

下面详细介绍QuickFinder

构造一个查找器

```
twaver.QuickFinder = function (dataBox, propertyName, propertyType, valueFunction, filterFunction)
```

dataBox：数据容器，查找器在此容器中查找

propertyName：属性名，如上面的例子查找"name"属性

propertyType：属性类型，默认为"accessor"，表示可直接获取的属性，如上面的"name"属性，对应的获取方式是data.getName()

属性类型支持三种：

accessor：直接，类似java中的javaBean属性类型，通过data[propertyName]读取

client：客户属性，通过data.getClient(propertyName)读取

style：样式属性，通过data.getStyle(propertyName)读取

valueFunction：值获取函数，默认根据上面三种方式获取数据，用户也可以指定值的获取方式，通过设置valueFunction实现

filterFunction：过滤函数，用于控制那些元素参与查找

函数体如下，传入data，返回true/false，表示这个元素是否参与查找
function(data){}

如何查找

查找器提供两个查找函数:

```
findFirst : function(value)
find : function(value)
```

findFirst：返回第一个找到的元素

find：返回找到的所有元素集合

实例分析：

下面的例子，分别构建了"name"查找器和用户"NO"属性查找器

```
function init() {
    var box = new twaver.DataBox();

    for (var i = 0; i < 20; i++) {
        var data = new twaver.Data();
        data.setName("group-" + parseInt(i / 4));
        data.setClient("NO", i % 4);
        box.add(data);
    }

    var nameFinder = new twaver.QuickFinder(box, "name");
    var noFinder = new twaver.QuickFinder(box, "NO", "client");

    data = nameFinder.findFirst("group-1");
    log("nameFinder.findFirst(\"group-1\")", data);

    var datas = nameFinder.find("group-1");
    log("nameFinder.find(\"group-1\")", datas);

    data = noFinder.findFirst(1);
    log("noFinder.findFirst(1)", data);

    datas = noFinder.find(1);
    log("noFinder.find(1)", datas);
    var finder = new twaver.QuickFinder(box, "name");
    finder.find("")
}

function getDataDescription(data) {
    return "name:" + data.getName() + ", NO:" + data.getClient("NO");
}

function log(title, data) {
    console.log(title);
    if (data instanceof twaver.Data) {
        console.log(getDataDescription(data));
    } else if (data instanceof twaver.List) {
        data.forEach(function(item) {
            console.log(getDataDescription(item));
        });
    }
}
```

输出结果：

```
nameFinder.findFirst("group-1")
name:group-1, NO:0
nameFinder.find("group-1")
name:group-1, NO:0
name:group-1, NO:1
name:group-1, NO:2
name:group-1, NO:3
noFinder.findFirst(1)
name:group-0, NO:1
noFinder.find(1)
name:group-0, NO:1
name:group-1, NO:1
name:group-2, NO:1
name:group-3, NO:1
```

name:group-4, NO:1

选中模型 - SelectionModel

选中模型服务于DataBox，用于管理元素选中信息，元素的选中，清除选中都通过SelectionModel实现。如下面的API将databox中的data元素设定为选中状态：

```
databox.getSelectionModel().appendSelection(data);
```

视图组件可以有自己独立的选中模型，与其关联的数据模型相绑定，最常见的拓扑图中要将某个网元选中，可以通过下面的API调用实现：

```
network.getSelectionModel().appendSelection(element);
```

构建选中模型对象

默认DataBox中自带一个选中模型，可以通过databox.getSelectionModel()获得，视图对象中的默认就是 databox的选中模型，可通过view.getSelectionModel()获得，如：network.getSelectionModel()或者 tree.getSelectionModel()

DataBox和视图组件都可以设置用户自己的选中模型，这种情况需要首先构建用户自己的选中模型实例

```
var mySelectionModel = new twaver.SelectionModel(databox);
```

构造参数传入需要绑定的databox即可
设置到databox 或者视图组件

```
databox.setSelectionModel(mySelectionModel);
network.setSelectionModel(mySelectionModel);
...
```

默认视图组件的选中模型即为databox的选中模型，这样可以保证同一个databox关联的视图组件同步选中，构建自己的选中模型通常用于某个视图组件需要独立选中的情况

如何选中元素和取消选中

SelectionModel提供下面的，实现追加选中，选中，全选，取消元素选中，取消所有元素的选中等功能

```
//追加选中元素，传入参数可以是单个元素，也可以是元素集合
appendSelection : function(datas)

//设置选中元素，与追加选中不同，此方法会先清除原始选中状态
setSelection : function(datas)

//选中databox中所有元素
selectAll : function()

//取消元素选中状态，传入参数可以是单个元素，也可以是元素集合
removeSelection : function(datas)

//清除所有元素的选中状态
```

```
clearSelection : function()
```

获取元素选中信息

选中模型中可以获得元素的选中信息，如所有选中的元素，选中个数，以及判断指定元素是否选中等等

```
//获得选中元素集合，注意此方法返回的是SelectionModel内部选中元素集合对象的引用，
//直接对这个集合操作会影响选中模型，所以不要对这个集合做修改操作
getSelection : function()

//获取当前选中元素集合，注意此方法与上个函数有区别，此方法返回的是新构建的集合类，
//而不是SelectionModel中原始的选中元素集合对象引用
//matchFunction : 匹配函数，传入IData，返回true或false，false表示排除
toSelection: function (matchFunction, scope)

//选中数量
size: function ()

//元素是否选中
contains: function (data)

//选中集合中的最后一个元素
getLastData: function ()

//选中集合中的第一个元素
getFirstData: function ()

//是否允许选中
isSelectable: function (data)
```

选中变化事件

选中模型管理元素的选中状态，当选中状态发生变化时，派发相应的选中变化事件，如调用追加元素选中函数会派发类型为"append"的选中变化事件

TWaver中可以监听选中变化事件，可以通过下面的函数使用

```
selectionModel.addSelectionChangeListener(function(e){
    console.log("Kind: " + e.kind + ", datas: " + e.datas.toString());
});
```

事件对象中包含两个属性"kind", "datas"，分别代表选中事件类型，事件数据，其中选中变化事件有五种子类型: append, set, remove, all, clear，分别对应选中模型上的五种操作元素选中状态的函数(appendSelection, setSelection, removeSelection, selectAll, clearSelection)

选择模式

选中模型提供三种选择模式("multipleSelection", "singleSelection", "noneSelection")，多选，单选，不可选，用于控制选中效果，也为视图组件选择模式提供了数据层支持，默认为多选模式

修改选择模式可通过下面的方法

```
selectionModel.setSelectionMode("singleSelection");
```

注意，当选择模式从多选切换为单选或者不可选时，TWaver内部会首先调用清除所有元素的选中状态

实例分析

下面的例子演示了元素选中操作，选中变化事件以及切换选择模式

```
function init() {
    var databox = new twaver.DataBox();
    var selectionModel = databox.getSelectionModel();

    for (var i = 0; i < 10; i++) {
        var data = new twaver.Data(i);
        data.setName("data_" + i);
        databox.add(data);
    }

    selectionModel.addSelectionChangeListener(function(e) {
        console.log("Kind: " + e.kind + ", datas: "
            + e.datas.toString());
    });

    selectionModel.appendSelection(databox.getDataById(0));
    selectionModel.appendSelection(databox.getDataById(1));
    selectionModel.removeSelection(databox.getDataById(0));
    selectionModel
        .setSelection([databox.getDataById(2), databox.getDataById(6)]);
    selectionModel.appendSelection([databox.getDataById(2),
        databox.getDataById(3), databox.getDataById(4)]);

    // single selection mode
    console.log("设置单选模式，首先清除当前的选中状态，以后每次只选择最多一个元素");
    selectionModel.setSelectionMode("singleSelection");
    selectionModel.appendSelection([databox.getDataById(2),
        databox.getDataById(3), databox.getDataById(4)]);
    console.log("selection size: " + selectionModel.size());

    console.log("设置不可选模式，会清除当前的选中状态");
    selectionModel.setSelectionMode("noneSelection");
    selectionModel.appendSelection([databox.getDataById(2),
        databox.getDataById(3), databox.getDataById(4)]);

    console.log("默认是多选模式");
    selectionModel.setSelectionMode("multipleSelection");

    console.log("\n设置过滤器，所有id大于5的都不可选");
    selectionModel.setFilterFunction(function(data) {
        return data.getId() > 5;
    });
    selectionModel.selectAll();
}
```

输出结果：

```
Kind: append, datas: data_0
Kind: append, datas: data_1
Kind: remove, datas: data_0
Kind: set, datas: data_1,data_2,data_6
Kind: append, datas: data_3,data_4
设置单选模式，首先清除当前的选中状态，以后每次只选择最多一个元素
Kind: clear, datas: data_2,data_6,data_3,data_4
Kind: append, datas: data_4
selection size: 1
33
设置不可选模式，会清除当前的选中状态
```

Kind: clear, datas: data_4

默认是多选模式

设置过滤器, 所有id大于5的都不可选

Kind: all, datas: data_6,data_7,data_8,data_9

Network组件


本章将详细介绍拓扑图组件的设计和使用，包括拓扑图的层次结构，背景的添加，各种过滤器的使用以及交互模式的切换：

- [Network层次结构](#)
- [Network过滤器的使用](#)
- [Network样式规则函数](#)
- [Network界面交互](#)

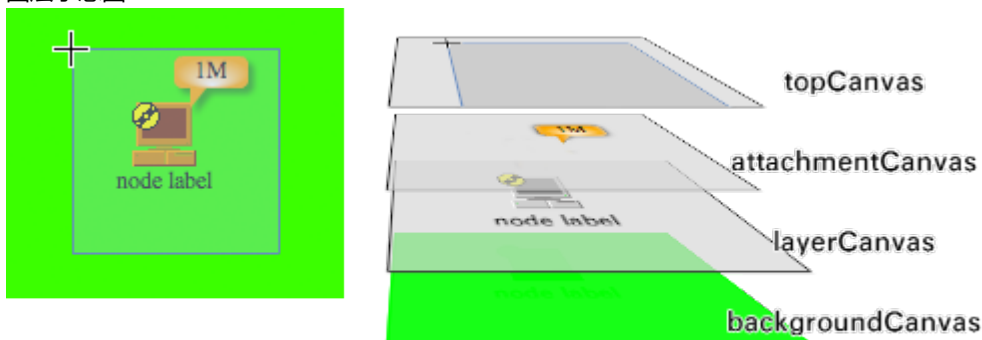
Network层次结构

拓扑图中所有的拓扑组件放在rootDiv中，有如下的层次结构

- rootDiv - - 主画布，所有拓扑组件都在其中
- topDiv - - 顶层画布，可用于绘制交互框选框，调整大小时的拖拽块
- attachmentDiv - - 放置顶层附件组件
- layerDiv - - 所有网元视图所在层
- layer n
- layer ...
- default layer
- bottomDiv - - 底层画布，可用于在背景之上绘制底纹

 对于网元附件（Attachment），默认添加在网元视图组件（ElementUI）之下，showInAttachmentDiv属性为true时，放置在attachmentDiv层

图层示意图：



Network中各图层面板都可以得到，用户可以在其中添加其他组件：

```
geRootDiv : function()
getTopDiv : function()
getAttachmentDiv : function()
getLayerDiv : function()
getBottomDiv : function()
```

下面的例子将在上面的图层中添加按钮，加深对这些层次的理解：

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init() {
      var network = new twaver.network.Network();
      var box = network.getElementBox();

      var node = new twaver.Node();
      node.setLocation(20,20);
      box.add(node);
      var node2 = new twaver.Node();
      node2.getAlarmState().setNewAlarmCount(twaver.AlarmSeverity.MAJOR, 1);
      node2.setLocation(150,50);
      box.add(node2);
    }
  </script>
</head>
</html>
```

```

var link = new twaver.Link(node, node2);
box.add(link);

var buttonInRootDiv = createDiv("div in rootDiv", 20, 90, 255, 100, 100);
network.getRootDiv().appendChild(buttonInRootDiv);

var buttonInTopDiv = createDiv("div in topDiv", 70, 115, 100, 255, 100);
network.getTopDiv().appendChild(buttonInTopDiv);

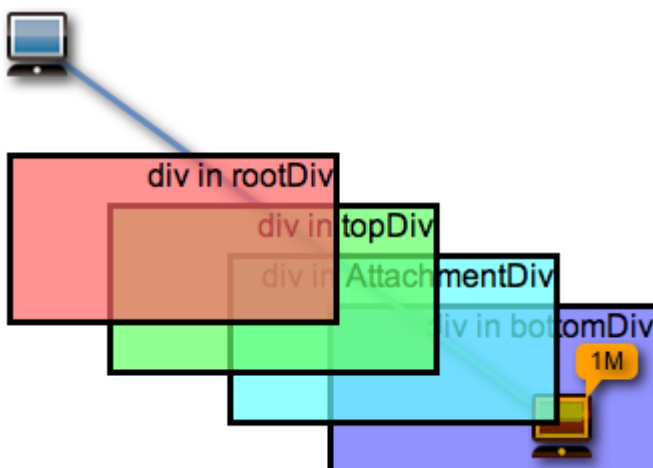
var buttonInAttachmentDiv = createDiv("div in AttachmentDiv", 130, 140, 100, 255, 255);
network.getAttachmentDiv().appendChild(buttonInAttachmentDiv);

var buttonInBottomDiv = createDiv("div in bottomDiv", 180, 165, 100, 100, 255);
network.getBottomDiv().appendChild(buttonInBottomDiv);

var networkDom = network.getView();
networkDom.style.width = "100%";
networkDom.style.height = "100%";
document.body.appendChild(networkDom);
}

function createDiv(text, x, y, r, g, b){
    var div= document.createElement("div");
    div.innerHTML = text;
    div.style.position = "absolute";
    div.style.textAlign = 'right';
    div.style.fontSize = "16px";
    div.style.left = x + "px";
    div.style.top = y + "px";
    div.style.border = "solid";
    div.style.backgroundColor = "rgba(" + r + ", " + g + ", " + b + ", 0.7)";
    div.style.height = "80px";
    div.style.width = "160px";
    return div;
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```



Network过滤器的使用

延续TWaver的一贯风格，TWaver HTML5提供了一系列过滤器，包括可见过滤器，可移动过滤器，可编辑过滤器，通过设置过滤器可以实现同一数据模型，不同信息的显示以及不同的操作模式，通常用于按用户权限或者网元类型设置不同的交互和视图。

Network中的过滤器包括

```
//可见过滤器  
get/setVisibleFunction : function(filter)  
//可移动过滤器  
get/setMovableFunction : function(filter)  
//可编辑过滤器  
get/setEditableFunction : function(filter)
```

过滤器使用示例，注意传入参数类型是Element，返回参数为Boolean

下面的示例表示包含孩子节点的网元可见

```
network.setVisibleFunction(function(node){  
    return node.getChildrenCount() > 0;  
});
```

Network样式规则函数

TWaver HTML5中使用规则函数的方式设置组件的样式，包括树图或拓扑图中节点颜色渲染，边框，冒泡等。本章介绍拓扑图中的相关规则。

拓扑图中网元的颜色渲染，边框颜色，告警颜色，告警文本，连线捆绑文本，工具条提示文本等都通过规则函数的方式实现，用户也可以设置自己的规则函数实现定制

Network中的样式规则：

getLabel：网元文本标签函数，用于设置网元的标签

getToolTip：工具条提示

getInnerColor：网元内渲染色函数

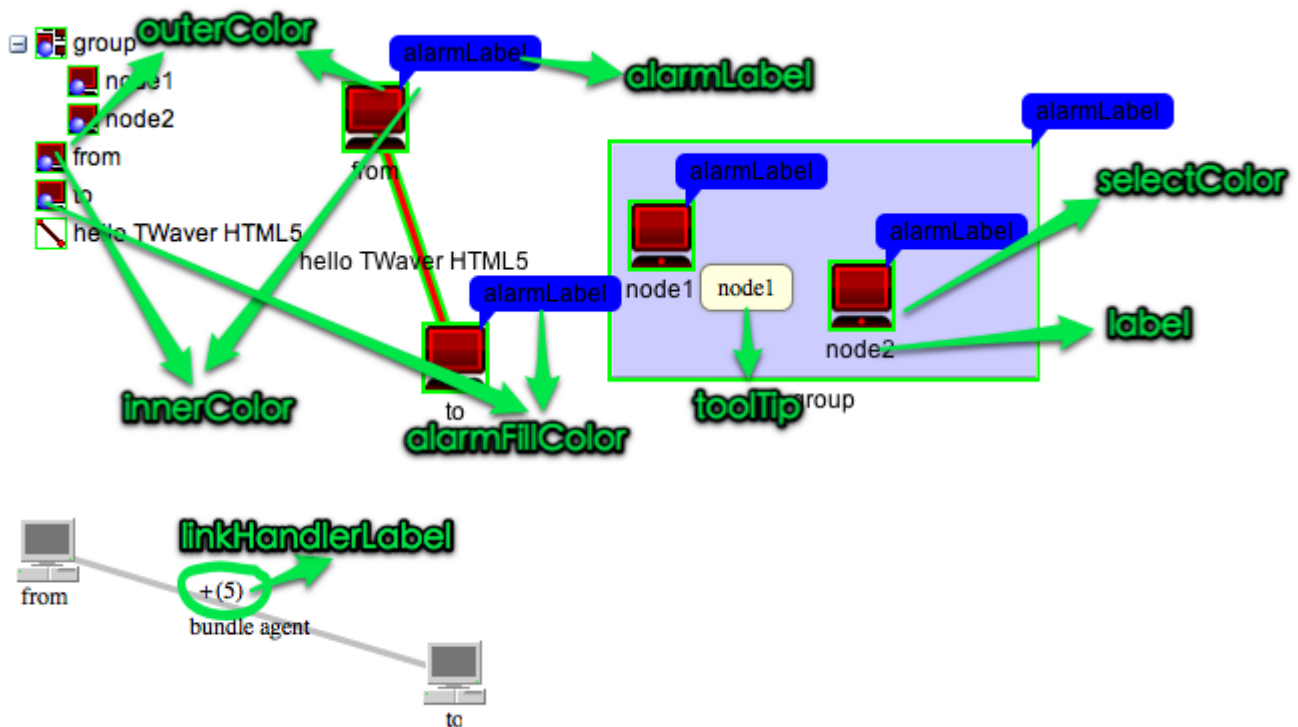
getOuterColor：网元边框颜色

getSelectColor：网元选中颜色

getAlarmFillColor：告警冒泡颜色

getAlarmLabel：告警冒泡文本

getLinkHandlerLabel：绑定连线文本



默认实现

TWaver的默认实现列举如下：

```
//label
Network#getLabel: function (data) {
    return data.getStyle('network.label') || data.getName();
};
```

```

Tree#getLabel = function (data) {
    return data.getName();
};
//toolTip
Network#getToolTip = function (data) {
    return data.getToolTip();
};

//innerColor
Network/Tree#
getInnerColor = function (data) {
    if (data.IElement) {
        var severity = data.getAlarmState().getHighestNativeAlarmSeverity();
        if (severity) {
            return severity.color;
        }
        return data.getStyle('inner.color');
    }
    return null;
};

//outerColor
Network/Tree#
getOuterColor = function (data) {
    if (data.IElement) {
        var severity = data.getAlarmState().getPropagateSeverity();
        if (severity) {
            return severity.color;
        }
        return data.getStyle('outer.color');
    }
    return null;
};

//selectColor
Network#getSelectColor: function (element) {
    return element.getStyle('select.color');
};

//alarmFillColor
Network/Tree#
getAlarmFillColor = function (data) {
    if (data.IElement) {
        var severity = data.getAlarmState().getHighestNewAlarmSeverity();
        if (severity) {
            return severity.color;
        }
    }
    return null;
};

//alarmLabel
getAlarmLabel: function (element) {
    var severity = element.getAlarmState().getHighestNewAlarmSeverity();
    if (severity) {
        var label = element.getAlarmState().getNewAlarmCount(severity) + severity.nickName;
        if (element.getAlarmState().hasLessSevereNewAlarms()) {
            label += "+";
        }
        return label;
    }
    return null;
}

//linkHandlerLabel
Network#

```

```
getLinkHandlerLabel: function (link) {
    if (link.isBundleAgent()) {
        return "(" + link.getBundleCount() + ")";
    }
    return null;
};
```

定制规则

下面的例子实现了几种规则函数的定制，代码如下：

```
<!DOCTYPE html>
<html>
<head>
    <title>TWaver HTML5 Demo</title>
    <script type="text/javascript" src="../demo/twaver.js"></script>
    <script type="text/javascript">
var box;
var number;
function init() {
    number = 0;
    var network = new twaver.network.Network();
    network.setToolTipEnabled(true);
    box = network.getElementBox();
    var tree = new twaver.controls.Tree(box);

    var group = new twaver.Group();
    group.setName("group");
    box.add(group);
    group.addChild(createTWaverNode("node1", 200, 100));
    group.addChild(createTWaverNode("node2", 300, 130));
    group.setExpanded(true);

    var from = createTWaverNode("from", 30, 30);
    var to = createTWaverNode("to", 70, 150);
    var link = new twaver.Link(from, to);
    link.setName("hello TWaver HTML5");
    box.add(link);

    // 树节点自身渲染颜色
    tree.getInnerColor = function(data) {
        return "#ff0000";
    };

    // 树节点边框颜色
    tree.getOuterColor = function(data) {
        return "#00ff00";
    };

    // 左下方小球的颜色，如果为null表示不显示
    tree.getAlarmFillColor = function(data) {
        if (data instanceof twaver.Element && !data.getAlarmState().isEmpty()) {
            return "#0000ff";
        }
        return null;
    };

    // network是同样的原理，同样的默认实现，
    // innerColor - 节点渲染色
    // outerColor - 边框颜色
    // alarmFillColor - 告警冒泡颜色，如果下面的告警文本为空，告警冒泡不显示
```

```

// alarmLabel - 告警文本
// Body 渲染色
network.getInnerColor = function(data) {
    return "#ff0000";
};

// 节点边框颜色
network.getOuterColor = function(data) {
    return "#00ff00";
};

// 告警冒泡的颜色, 如果下面相应的alarmLabel返回null或者颜色为null不显示
network.getAlarmFillColor = function(data) {
    if (data instanceof twaver.Element && !data.getAlarmState().isEmpty()) {
        return "#0000ff";
    }
    return null;
};

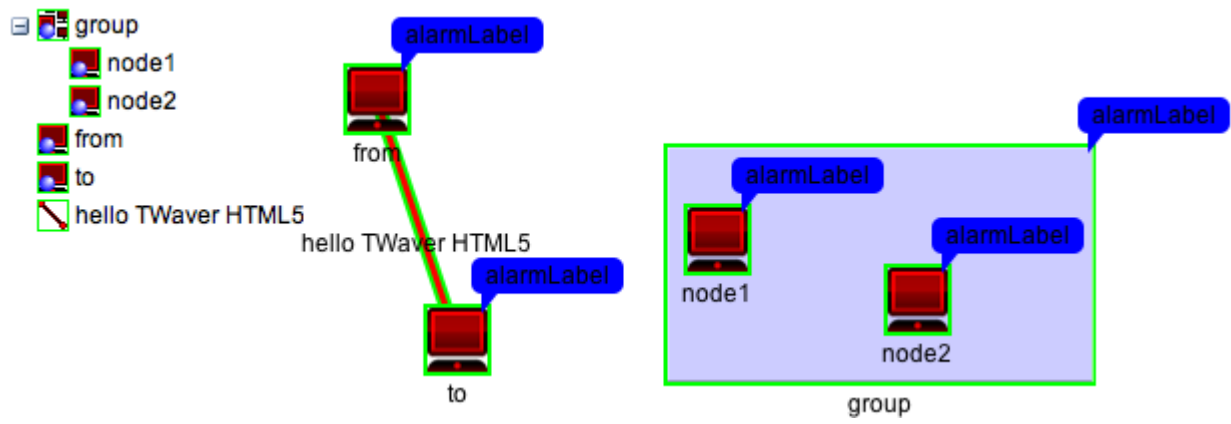
network.getAlarmLabel = function(element) {
    if (!element.getAlarmState().isEmpty()) {
        return "alarmLabel";
    }
    return null;
};

network.getSelectedColor = function(element) {
    return "#ffff00";
};

var treeDom = tree.getView();
treeDom.style.width = "150px";
treeDom.style.height = "100%";
var networkDom = network.getView();
networkDom.style.left = "150px";
networkDom.style.width = "100%";
networkDom.style.height = "100%";
document.body.appendChild(treeDom);
document.body.appendChild(networkDom);
}

function createTWaverNode(name, x, y) {
    var node = new twaver.Node();
    node.setName(name);
    node.setToolTip(name);
    node.getAlarmState().increaseNewAlarm(twaver.AlarmSeverity.MAJOR);
    node.setClient("number", number++);
    node.setLocation(x, y);
    box.add(node);
    return node;
}
</script>
</head>
<body onload="init()" style="margin:10;"> </body>
</html>

```

Network界面交互

InteractionHandler

拓扑图上的鼠标键盘交互通过一堆Interaction组成，每个Interaction中包含独立的事件监听和卸载操作，其中最基本的实现类是twaver.network.interaction.BaseInteraction，其他交互类都是从它继承衍生出来的，下面是*BaseInteraction{*}的实现代码，其中的setUp函数用于添加监听，tearDown函数卸载监听，实现交互监听类时，通常需要重写这两个函数

```
twaver.network.interaction.BaseInteraction = function (network) {
    this.network = network;
};
twaver.Util.ext('twaver.network.interaction.BaseInteraction', Object, {
    setUp: function () {

    },
    tearDown: function () {

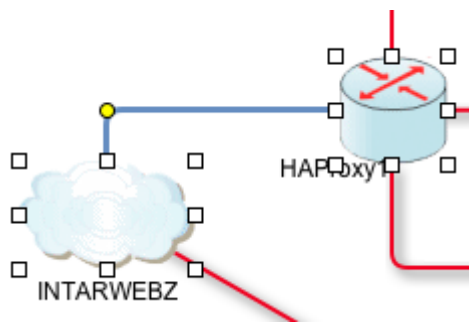
    },
    addListener: function () {
        for (var i = 0; i < arguments.length; i++) {
            var type = arguments[i];
            twaver.Util.addEventListener(type, 'handle_' + type, this.network.getView(), this);
        }
    },
    removeListener: function () {
        for (var i = 0; i < arguments.length; i++) {
            twaver.Util.removeEventListener(arguments[i], this.network.getView(), this);
        }
    }
});
```

常用交互模式

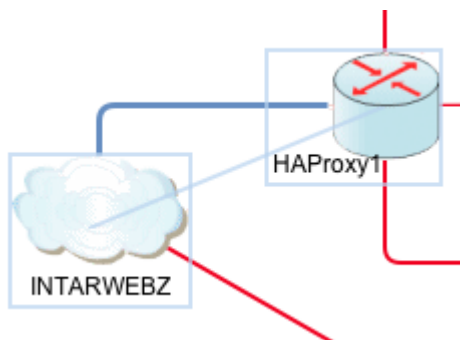
Network中预定义了常用的几种交互模式，包括默认模式，编辑模式，创建连线模式，创建多点连线，多边形网元以及触控交互模式，此外放大镜也很容易实现：

```
setDefaultInteractions: function (lazyMode)
setPanInteractions: function ()
setEditInteractions: function (lazyMode)
setCreateElementInteractions: function (type)
setCreateLinkInteractions: function (type)
setCreateShapeLinkInteractions: function (type)
setCreateShapeNodeInteractions: function (type)
setTouchInteractions: function ()
```

编辑模式



创建Link交互模式



创建ShapeLink交互模式



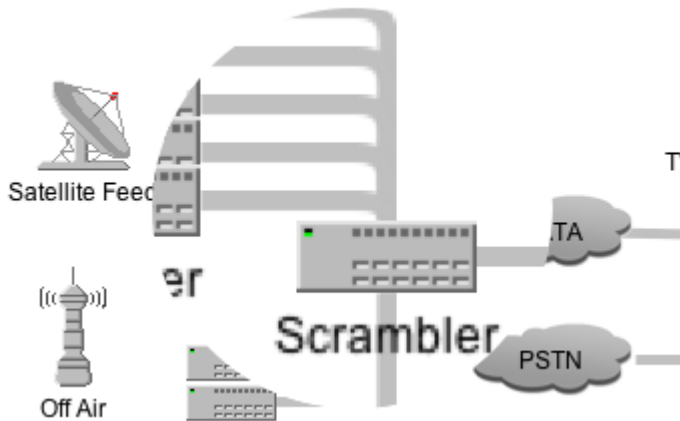
创建多边形交互模式



放大镜交互：

```
network.setInteractions([
  new twaver.network.interaction.SelectInteraction(network),
```

```
new twaver.network.interaction.MoveInteraction(network),
new twaver.network.interaction.MagnifyInteraction(network),
new twaver.network.interaction.DefaultInteraction(network));
```



下面的例子可以切换各种交互模式，并简单实现了拖拽按钮创建网元的功能，用户可以参考

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var network;
    function init(){
      network = new twaver.network.Network();

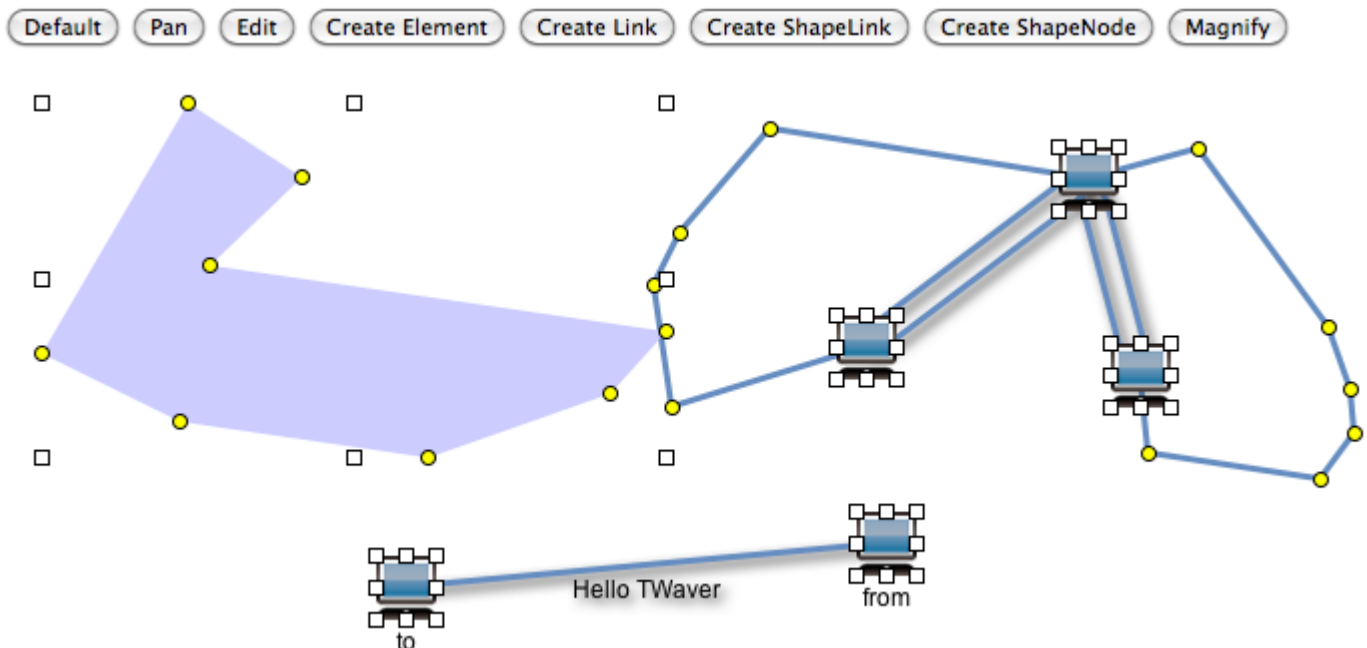
      var box = network.getElementBox();
      var node = new twaver.Node();
      node.setName("from");
      node.setLocation(100, 100);
      box.add(node);
      var node2 = new twaver.Node();
      node2.setName("to");
      node2.setLocation(300, 300);
      box.add(node2);

      var link = new twaver.Link(node, node2);
      link.setName("Hello TWaver");
      link.setToolTip("<b>Hello TWaver</b>");
      box.add(link);

      var networkDom = network.getView();
      networkDom.style.width = "100%";
      networkDom.style.height = "400px";
      document.getElementById("network").appendChild(networkDom);
    }
    function setMagnifyInteraction(){
      network.setInteractions([
        new twaver.network.interaction.SelectInteraction(network),
        new twaver.network.interaction.MoveInteraction(network),
        new twaver.network.interaction.MagnifyInteraction(network),
        new twaver.network.interaction.DefaultInteraction(network));
      ]
    }
  </script>
</head>
```

```
<body onload="init()" style="margin:0px;">
<div style="width: 100%;" >
<button onclick="network.setDefaultInteractions();">Default</button>
<button onclick="network.setPanInteractions();">Pan</button>
<button onclick="network.setEditInteractions();">Edit</button>
<button onclick="network.setCreateElementInteractions();">Create Element</button>
<button onclick="network.setCreateLinkInteractions();">Create Link</button>
<button onclick="network.setCreateShapeLinkInteractions();">Create ShapeLink</button>
<button onclick="network.setCreateShapeNodeInteractions();">Create ShapeNode</button>
<button onclick="setMagnifyInteraction();">Magnify</button>
</div>
<div id="network" />
</body>
</html>
```

运行界面



定制拓扑图交互模式

定制交互监听器需要继承BaseInteraction类，然后重写setUp和tearDown函数，分别执行添加鼠标键盘监听器和卸载这些监听的动作。

下面我们以一个实例来说明，我们将实现鼠标移动到网元上，网元高亮显示的功能。

高亮显示交互器

首先我们需要实现一个高亮交互监听器，这里继承BaseInteraction，实现如下：

```
function highlightInteraction(network) {
  twaver.network.interaction.MoveInteraction.superClass.constructor.call(
    this, network);
}
twaver.Util.ext('highlightInteraction', twaver.network.interaction.BaseInteraction,
  {
    setUp : function() {
      this.addListener('mousemove');
    },
    tearDown : function() {
      this.removeListener('mousemove');
    }
  });
```

```

        this.end();
    },
    handle_mousemove : function(e) {
        var element = this.network.getElementAt(e);
        if (element) {
            this.highlight(element);
        } else {
            this.reset();
        }
    },
    highlightElement : null,
    highlightColor : "#FF0000",
    oldColor : null,
    highlight : function(element) {
        this.reset();
        this.oldColor = element.getStyle("inner.color");
        this.highlightElement = element;
        element.setStyle("inner.color", this.highlightColor);
    },
    reset : function() {
        if (this.highlightElement) {
            this.highlightElement
                .setStyle('inner.color', this.oldColor);
        }
    }
});

```

这里我们监听了mousemove事件，在鼠标点位置有网元时，高亮显示这个网元，否则重置

使用定制的交互器

下面来使用上面定义的交互器，通过调用Network#setInteractions(array)函数，给拓扑图设置一组交互监听器，这样各个交互监听器都可以各司其职，选中监听器负责选中交互，移动交互器负责移动交互处理，同样前面定义的高亮交互器实现鼠标划过网元高亮显示的动作。

```

network.setInteractions([
    new twaver.network.interaction.SelectInteraction(network),
    new twaver.network.interaction.MoveInteraction(network),
    new highlightInteraction(network),
    new twaver.network.interaction.DefaultInteraction(network)]);

```

完整代码

```

<!DOCTYPE html>
<html>
<head>
    <title>TWaver HTML5 Demo </title>
    <script type="text/javascript" src="../demo/twaver.js"></script>
    <script type="text/javascript">
function highlightInteraction(network) {
    twaver.network.interaction.MoveInteraction.superClass.constructor.call(
        this, network);
}
twaver.Util.ext('highlightInteraction', twaver.network.interaction.BaseInteraction,
    {
        setUp : function() {
            this.addListener('mousemove');
        },
        tearDown : function() {
            this.removeListener('mousemove');

```

```

        this.end();
    },
    handle_mousemove : function(e) {
        var element = this.network.getElementAt(e);
        if (element) {
            this.highlight(element);
        } else {
            this.reset();
        }
    },
    highlightElement : null,
    highlightColor : "#FF0000",
    oldColor : null,
    highlight : function(element) {
        this.reset();
        this.oldColor = element.getStyle("inner.color");
        this.highlightElement = element;
        element.setStyle("inner.color", this.highlightColor);
    },
    reset : function() {
        if (this.highlightElement) {
            this.highlightElement
                .setStyle('inner.color', this.oldColor);
        }
    }
});
var network;
function init() {
    network = new twaver.network.Network();

    var box = network.getElementBox();
    var node = new twaver.Node();
    node.setName("from");
    node.setLocation(100, 100);
    box.add(node);
    var node2 = new twaver.Node();
    node2.setName("to");
    node2.setLocation(300, 300);
    box.add(node2);

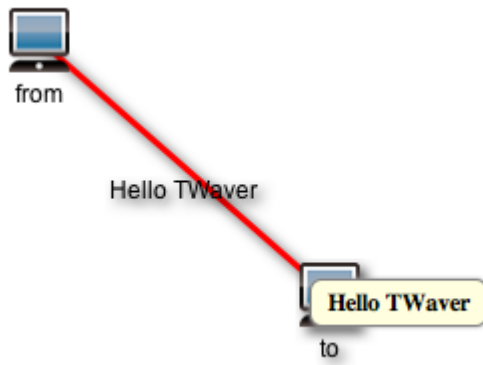
    var link = new twaver.Link(node, node2);
    link.setName("Hello TWaver");
    link.setToolTip("<b>Hello TWaver</b>");
    box.add(link);

    var networkDom = network.getView();
    networkDom.style.width = "100%";
    networkDom.style.height = "100%";
    document.body.appendChild(networkDom);

    network.setInteractions([
        new twaver.network.interaction.SelectInteraction(network),
        new twaver.network.interaction.MoveInteraction(network),
        new highlightInteraction(network),
        new twaver.network.interaction.DefaultInteraction(network)
    ]);
}
</script>
</head>
<body onload="init()" style="margin:0px;">
</body>
</html>

```

运行效果：



Tree组件

twaver.controls.Tree组件用于展示DataBox中数据元素的层次结构，树图上节点的父子关系由数据元素的父子关系决定，树节点的先后位置由数据元素在其父节点的位置所决定。

- [树节点样式](#)
- [树的层次与排序](#)
- [树的勾选模式](#)

创建一个树组件

树图组件的创建与拓扑图类似，也是通过API创建，构造参数可以传入databox，实现与这个数据容器的关联

```
var tree = new twaver.controls.Tree();
```

示例

下面我们创建一个树组件，通过父子关系设置树节点数据

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo </title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var tree = new twaver.controls.Tree();

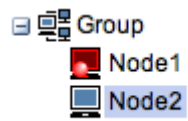
      var box = tree.getDataBox();
      var group = new twaver.Group();
      group.setName('Group');
      box.add(group);

      var node1 = new twaver.Node();
      node1.setName("Node1");
      node1.setParent(group);
      node1.getAlarmState().increaseNewAlarm(twaver.AlarmSeverity.CRITICAL);
      box.add(node1);
      var node2 = new twaver.Node();
      node2.setName("Node2");
      node2.setParent(group);
      box.add(node2);
      tree.getSelectionModel().setSelection(node2);

      var treeDom = tree.getView();
      treeDom.style.width = "100%";
      treeDom.style.height = "100%";
      document.body.appendChild(treeDom);

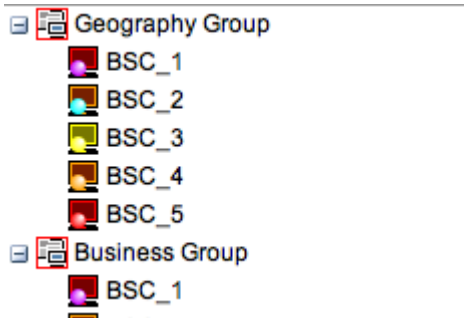
      tree.expandAll();
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

运行界面

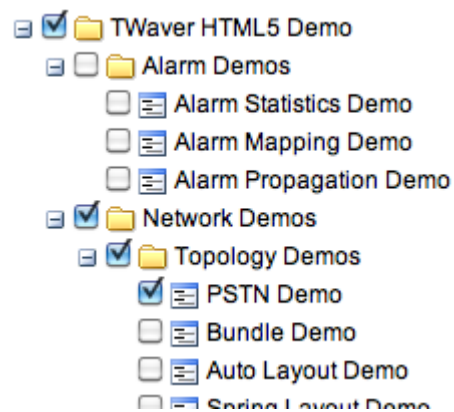


树图的呈现效果

树节点告警效果



勾选框模式的树图



树节点样式

TWaverHTML5的树节点由图标+文字组成，其中图标支持染色和冒泡挂载的功能

树节点样式

设置图标

```
□data.setIcon('iconName');
```

设置节点标签

```
□data.setName('001');
```

树节点样式示例

下面的例子展示了树图的一般使用，包括icon的设置，标签的定制，以及告警渲染

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var box = new twaver.ElementBox();
      var tree = new twaver.controls.Tree(box);

      var group = new twaver.Group();
      group.setName('Group');
      box.add(group);

      for(var i = 0; i < 3; i++){
        var node1 = new twaver.Node();
        node1.setIcon("images/twaver-small.png");
        node1.setName("Node-" + i);
        node1.setParent(group);
        box.add(node1);
        for(var j = 0; j < 3; j++){
          var node2 = new twaver.Node();
          node2.setName("Node-" + i + "-" + j);
          node2.getAlarmState().setNewAlarmCount(twaver.AlarmSeverity.MAJOR, 1);
          node2.setParent(node1);
          box.add(node2);
        }
      }

      var treeDom = tree.getView();
      treeDom.style.width = "100%";
      treeDom.style.height = "100%";
      document.body.appendChild(treeDom);

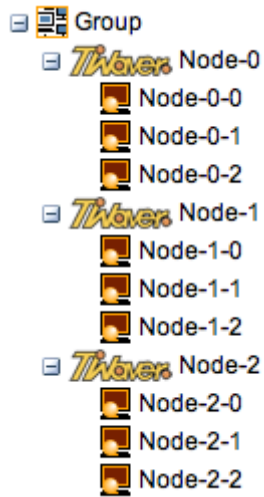
      tree.expandAll();
    }
  </script>
</head>
</html>
```

```

</script>
</head>
<body onload="init()" style="margin:20;">
</body>
</html>

```

呈现如下：



树的层次与排序

树节点的层次结构由节点的父子关系，以及孩子的次序所决定，我们可以改变节点的父节点，也可以调整孩子的先后次序，从而实现树结构的调整

层次按网元父子关系

```
□node.setParent(parent);  
□node.addChild(child);
```

同层节点顺序

□默认按加入到DataBox的先后顺序□可以通过调整网元在dataBox中的顺序来更改树节点的次序

```
□box.move***(node);
```

设置比较器进行排序

```
□tree.setSortFunction(compareFunction);
```

比如按名称降序，可以使用下面的代码

```
tree.setSortFunction(function(d1, d2){  
    return d1.getName() > d2.getName() ? -1 : 1;  
});
```

排序示例

下面的例子，我们将创建多个随机名称的树节点，并对其进行降序排序

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>TWaver HTML5 Demo</title>  
  <script type="text/javascript" src="../demo/twaver.js"></script>  
  <script type="text/javascript">  
    function init(){  
      var tree = new twaver.controls.Tree();  
  
      var box = tree.getDataBox();  
  
      for(var i = 0; i < 3; i++){  
        var node1 = new twaver.Node();  
        node1.setName("Node - " + Math.floor(Math.random()*10));  
        box.add(node1);  
        for(var j = 0; j < 3; j++){  
          var node2 = new twaver.Node();  
          node2.setName("Child - " + Math.floor(Math.random()*10));  
          node2.setParent(node1);  
        }  
      }  
    }  
  </script>  
</head>  
</html>
```

```

        box.add(node2);
    }
}

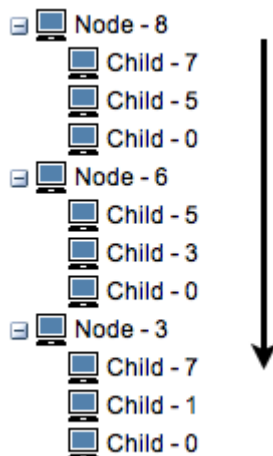
tree.setSortFunction(function(d1, d2){
    return d1.getName() > d2.getName() ? -1 : 1;
});

var treeDom = tree.getView();
treeDom.style.width = "100%";
treeDom.style.height = "100%";
document.body.appendChild(treeDom);

tree.expandAll();
}
</script>
</head>
<body onload="init()" style="margin:20;">
</body>
</html>

```

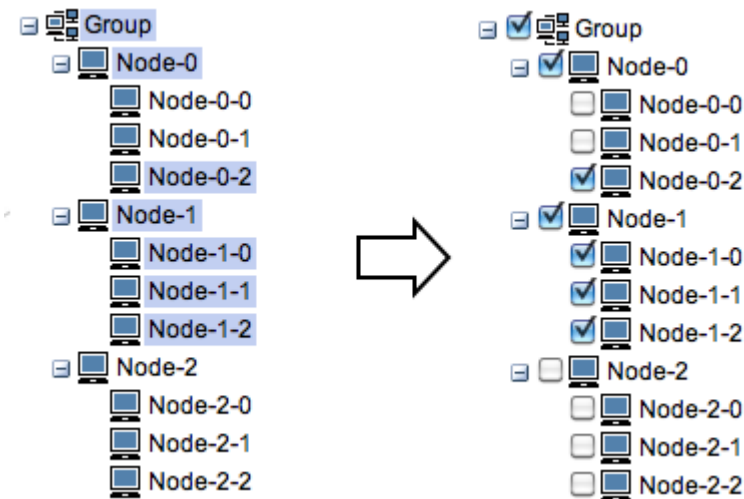
运行界面



树的勾选模式

TWaver HTML5的树图组件支持勾选模式，通过树节点前面的勾选框表示节点的选中状态。

默认模式到勾选模式



勾选模式

树图提供四种勾选模式，通过下面的方法切换：

```
Tree.setCheckMode(...);
```

默认勾选模式 - default

所有节点都可以勾选

孩子勾选模式 - children

所有的分支节点可以勾选

子孙勾选模式 - descendant

父节点被勾选，则其子孙节点自动被勾选

子孙祖先勾选模式 - descendantAncestor

父节点被勾选，则其子孙节点自动被勾选，子孙节点被勾选，则其祖先节点也自动勾选

下面是勾选模式树图的使用示例

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var tree = new twaver.controls.Tree();
```

```

var box = tree.getDataBox();
var group = new twaver.Group();
group.setName('Group');
box.add(group);

for(var i = 0; i < 3; i++){
    var node1 = new twaver.Node();
    node1.setName("Node-"+i);
    node1.setParent(group);
    box.add(node1);
    for(var j = 0; j < 3; j++){
        var node2 = new twaver.Node();
        node2.setName("Node-"+ i + "-" + j);
        node2.setParent(node1);
        box.add(node2);
    }
}

tree.setCheckMode("descendantAncestor");

var treeDom = tree.getView();
treeDom.style.width = "100%";
treeDom.style.height = "100%";
document.body.appendChild(treeDom);

tree.expandAll();
}
</script>
</head>
<body onload="init()" style="margin:20;">
</body>
</html>

```


Table组件

twaver.controls.Table组件实现了与DataBox的数据绑定，用以展示容器内元素属性信息，每行对应一个数据元素，本章将介绍表格组件的使用，表格列的定义，过滤以及排序功能的使用。

- [表格列的设置](#)
- [表格数据排序](#)

下面的示例中Table组件显示Node的name,id, icon属性

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var table = new twaver.controls.Table();

      var box = table.getDataBox();
      var root = new twaver.Data();
      root.setIcon("group_icon");
      root.setName('Root');
      box.add(root);

      var i = 100;
      while(i-->0){
        data = new twaver.Data();
        data.setName("TWaver-" + i);
        data.setParent(root);
        box.add(data);
      }

      createColumn(table,'Name', 'name', 'accessor', 'string');
      createColumn(table,'Id', 'id', 'accessor', 'string');
      createColumn(table,'Icon', 'icon', 'accessor');

      var tablePane = new twaver.controls.TablePane(table);
      var tableDom = tablePane.getView();
      tableDom.style.width = "100%";
      tableDom.style.height = "100%";
      document.body.appendChild(tableDom);
      window.onresize = function(){
        tablePane.invalidate();
      }
    }

    function createColumn(table, name, propertyName, propertyType, valueType) {
      var column = new twaver.Column(name);
      column.setName(name);
      column.setPropertyName(propertyName);
      column.setPropertyType(propertyType);
      if (valueType) column.setValueType(valueType);
      table.getColumnBox().add(column);
      return column;
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

运行界面：

Name	Id	Icon	
Root	C5597E4FB7D94A209666B2F50...	group_icon	
TWaver-99	94D22BF7B54E444A91EFF430...	data_icon	
TWaver-98	BEC49F6B3EE2449A817D8995...	data_icon	
TWaver-97	AAE7AFC73DC946C9981F1FBC...	data_icon	
TWaver-96	6AE75956CC5F421D8BC2B70A...	data_icon	
TWaver-95	74AFFD229557407CB AFF100B...	data_icon	
TWaver-94	C46BCA2BF2474DA69BD652B0...	data_icon	
TWaver-93	DDC280D6973A44638D17C3C5...	data_icon	
TWaver-92	FF5C11F51A994A6D8273361C4...	data_icon	

表格列的设置

TWaver HTML5中的表格组件用于展示DataBox中元素属性，每行对应一个数据元素，每一列对应元素的一种属性。表格列的实现类是twaver.Column，首先我们需要设置该列关联数据元素的哪个属性，余下还可以设置列头名称，列头呈现方式，单元格如何呈现等等

表格列设置

下面是一个最简单的列设置，显示数据元素的name属性

```
var column = new twaver.Column();
column.setName("Name");//设置列头名称
column.setPropertyName("name");//设置属性名
column.setPropertyType("accessor");//设置属性类型，默认就是"accessor"类型
column.setValueType("string");//设置值类型，默认就是"string"类型
table.getColumnBox().add(column);//设置到表格列容器
```

上面的代码中，我们设置了列头名称，属性名称，属性类型和属性值类型，其中属性类型有四种：field、accessor、style、client，不同的属性类型有不同的读取方式：
field属性：指node.name这样的属性；
accessor属性：node.getName(),node.setName(...);
style属性：node.getStyle("name"),node.setStyle("name", ...);
client属性：node.getClient("name"), node.setClient("name", ...)。

再说一个style属性列的例子，定义一列显示数据元素的"inner.color"样式属性，并设置不可编辑

```
column = new twaver.Column();
column.setName("Inner Color");
column.setPropertyName("inner.color");
column.setPropertyType("style");
column.setValueType("color");
table.getColumnBox().add(column);
```

完整示例

下面我看看完整的示例代码

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript" src="../demo/demo.js"></script>
  <script type="text/javascript">
    function init(){
      var table = new twaver.controls.Table();
      var box = table.getDataBox();

      var i = 100;
      while(i-->0){
        data = new twaver.Node();
        data.setName("TWaver-" + i);
        data.setStyle("inner.color", demo.Util.randomColor());
        box.add(data);
      }
    }
  </script>
</head>
</html>
```

```







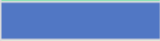

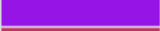

var column = new twaver.Column();
column.setName("Name");
column.setPropertyName("name");
column.setPropertyType("accessor");
column.setValueType("string");
table.getColumnBox().add(column);

column = new twaver.Column();
column.setName("Inner Color");
column.setPropertyName("inner.color");
column.setPropertyType("style");
column.setValueType("color");
table.getColumnBox().add(column);

var tablePane = new twaver.controls.TablePane(table);
var tableDom = tablePane.getView();
tableDom.style.width = "100%";
tableDom.style.height = "100%";
document.body.appendChild(tableDom);
window.onresize = function(){
    tablePane.invalidate();
}
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```

运行界面

Name	Inner Color
TWaver-99	
TWaver-98	
TWaver-97	
TWaver-96	
TWaver-95	
TWaver-94	
TWaver-93	
TWaver-92	
TWaver-91	
TWaver-90	

表格数据排序

调整数据顺序

表格中的数据可以通过调用dataBox.move***(element)来作调整，实现表格行的上下移动。

表格列排序

此外表格组件的每一列都支持排序功能，默认点击列头切换排序状态：升序，降序，不排序，默认的排序比较器使用字符串比较，特殊数据类型的可以定制排序比较器，下面我们将以颜色亮度比较为例，介绍表格排序的使用。排序比较器设置在表格列(Column)上，使用方法Column#setSortFunction(...);
如：

```
var column = new twaver.Column();
column.setName("Inner Color");
column.setPropertyName("inner.color");
column.setPropertyType("style");
column.setValueType("color");
column.setSortFunction(function(color1, color2) {
    if (!color1) {
        return -1;
    }
    if (!color2) {
        return 1;
    }
    var number1 = parseInt(color1.substring(1), 16);
    var r1 = (number1 >> 16) & 0xFF;
    var g1 = (number1 >> 8) & 0xFF;
    var b1 = number1 & 0xFF;
    var number2 = parseInt(color2.substring(1), 16);
    var r2 = (number2 >> 16) & 0xFF;
    var g2 = (number2 >> 8) & 0xFF;
    var b2 = number2 & 0xFF;
    return (r1 + g1 + b1) - (r2 + g2 + b2);
});
table.getColumnBox().add(column);
```

这里颜色默认为字符串类型，格式如："#FF0000"，首先我们需要转出rgb数值，然后使用r+g+b的值作为排序比较的值。

完整示例

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript" src="../demo/demo.js"></script>
  <script type="text/javascript">
    function init(){
      var table = new twaver.controls.Table();
      var box = table.getDataBox();

      var i = 100;
      while(i-->0){
        data = new twaver.Node();
        data.setName("TWaver-" + i + "-" + i%10);
        data.setStyle("inner.color", demo.Util.randomColor());
      }
    }
  </script>
</head>
<body>
  <table>
    <tr>
      <th>Name</th>
      <th>Color</th>
    </tr>
    <tr>
      <td>TWaver-99-9</td>
      <td>#FF0000</td>
    </tr>
    <tr>
      <td>TWaver-98-8</td>
      <td>#00FF00</td>
    </tr>
    <tr>
      <td>TWaver-97-7</td>
      <td>#0000FF</td>
    </tr>
    <tr>
      <td>TWaver-96-6</td>
      <td>#FF00FF</td>
    </tr>
    <tr>
      <td>TWaver-95-5</td>
      <td>#00FFFF</td>
    </tr>
    <tr>
      <td>TWaver-94-4</td>
      <td>#FFFF00</td>
    </tr>
    <tr>
      <td>TWaver-93-3</td>
      <td>#FF00FF</td>
    </tr>
    <tr>
      <td>TWaver-92-2</td>
      <td>#0000FF</td>
    </tr>
    <tr>
      <td>TWaver-91-1</td>
      <td>#00FF00</td>
    </tr>
    <tr>
      <td>TWaver-90-0</td>
      <td>#FF0000</td>
    </tr>
  </table>
</body>
</html>
```

```

        box.add(data);
    }

    var column = new twaver.Column();
    column.setName("Name");
    column.setPropertyName("name");
    column.setPropertyType("accessor");
    column.setValueType("string");
    table.getColumnBox().add(column);

    column = new twaver.Column();
    column.setName("Inner Color");
    column.setPropertyName("inner.color");
    column.setPropertyType("style");
    column.setValueType("color");
    column.setSortFunction(function(color1, color2){
        if(!color1){
            return -1;
        }
        if(!color2){
            return 1;
        }
        var number1 = parseInt(color1.substring(1, 16));
        var r1 = (number1 >> 16) & 0xFF;
        var g1 = (number1 >> 8) & 0xFF;
        var b1 = number1 & 0xFF;
        var number2 = parseInt(color2.substring(1, 16));
        var r2 = (number2 >> 16) & 0xFF;
        var g2 = (number2 >> 8) & 0xFF;
        var b2 = number2 & 0xFF;
        return (r1 + g1 + b1) - (r2 + g2 + b2);
    });
    table.getColumnBox().add(column);

    var tablePane = new twaver.controls.TablePane(table);
    var tableDom = tablePane.getView();
    tableDom.style.width = "100%";
    tableDom.style.height = "100%";
    document.body.appendChild(tableDom);
    window.onresize = function(){
        tablePane.invalidate();
    }
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```

运行界面：

Name	Inner Color
TWaver-76-6	
TWaver-42-2	
TWaver-30-0	
TWaver-19-9	
TWaver-4-4	
TWaver-22-2	

Name	Inner Color
TWaver-14-4	
TWaver-43-3	
TWaver-37-7	
TWaver-67-7	
TWaver-26-6	
TWaver-80-0	

Chart组件

TWaver HTML5 提供丰富的图表组件，其设计思路延续了TWaver视图组件的风格，按MVC设计模式，与dataBox数据容器关联，用图表的形式显示数据容器中的chart数据。

- [Chart Value & Chart Values](#)
- [BarChart](#)
- [LineChart](#)
- [PieChart](#)
- [DialChart](#)
- [BubbleChart](#)
- [RadarChart](#)

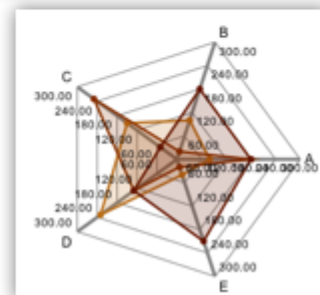
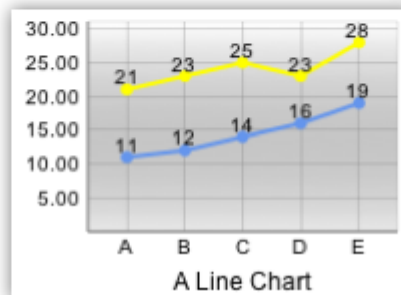
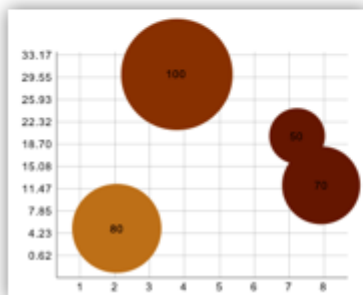
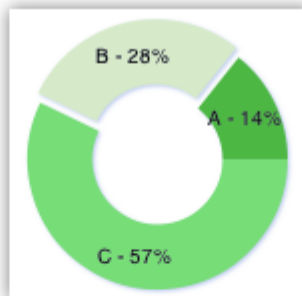
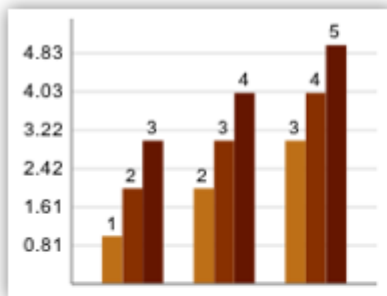


Chart Value & Chart Values

Chart与DataBox关联，以图表的形式表现dataBox中数据元素的Chart数值属性，数值分两种，chart value和 chart list values，分别表示单个图表值和多个图表值，通过网元的样式属性（chart.value, chart.values）进行设置

```
element.setStyle("chart.value", 27);
element.setStyle("chart.values", [27, 37, 48]);
```

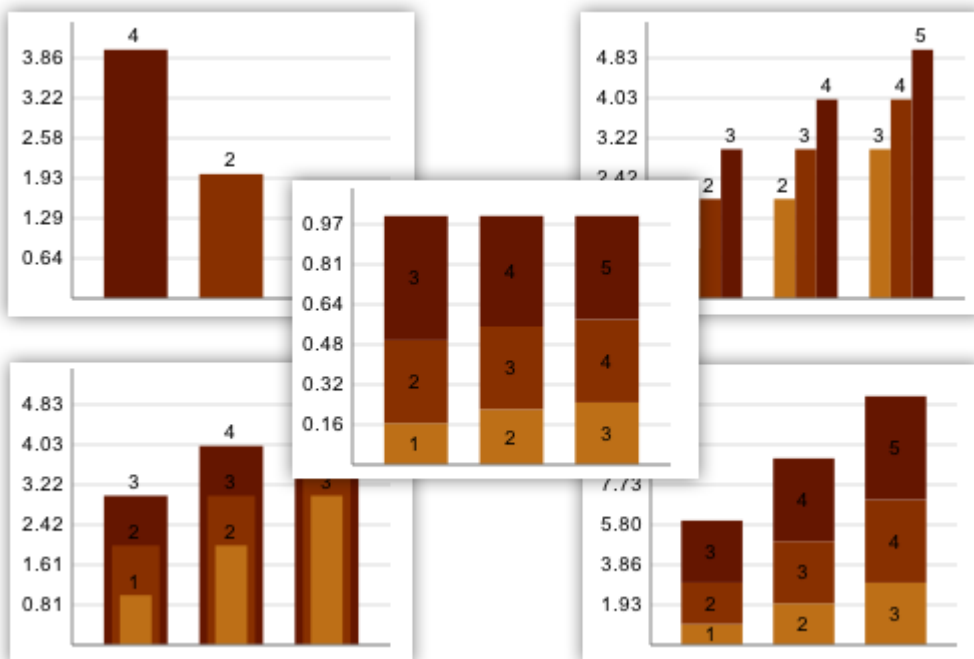
Chart	支持Chart Value	支持Chart Values
BarChart	True	True
LineChart	False	True
PieChart	True	False
DialChart	True	False
BubbleChart	False	True
RadarChart	False	True

BarChart

柱状图有多种显示模式，默认模式时每个柱对应一个数据元素，其高度由该元素的chart值所决定。此外还有分组、叠加、嵌套、百分比模式，设置柱状图显示模式如下：

```
chart.setType("group");
```

- "default" - 默认模式
- "group" - 分组模式
- "stack" - 叠加模式
- "layer" - 嵌套模式
- "percent" - 百分比模式



示例：

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init() {
      var chart = new twaver.charts.BarChart();
      var box = chart.getDataBox();

      // chart.setType("group");
      chart.setType("stack");
      // chart.setType("layer");
      // chart.setType("percent");

      var data = new twaver.Element();
      data.setStyle("chart.value", 4);
      data.setStyle("chart.values", [3,4,5]);
      data.setStyle("chart.color", "#651600");
```

```

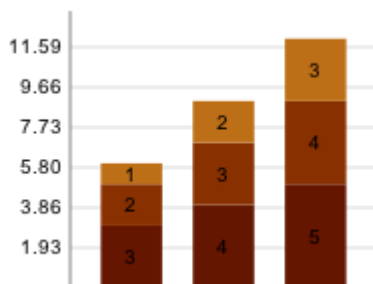
box.add(data);
data = new twaver.Element();
data.setStyle("chart.value", 2);
data.setStyle("chart.values", [2,3,4]);
data.setStyle("chart.color", "#883000");
box.add(data);
data = new twaver.Element();
data.setStyle("chart.value", 1);
data.setStyle("chart.values", [1,2,3]);
data.setStyle("chart.color", "#BD6F17");
box.add(data);

var chart = chart.getView();
chart.style.width = "200px";
chart.style.height = "150px";
document.body.appendChild(chart);

chart = chart.getView();
chart.style.width = "200px";
chart.style.height = "150px";
document.body.appendChild(chart);
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```

运行界面



LineChart

线图是最常用的图表类型之一，用线条表现趋势走向，TWaver HTML5 中的线图每条线表示一个数据元素，线上的一组点对应该数据元素的[chart values](#)

示例

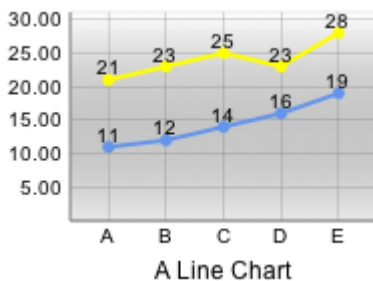
```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var box;
    function init() {
      var chart = new twaver.charts.LineChart();
      box = chart.getDataBox();

      var data = new twaver.Element();
      data.setStyle("chart.values", [11, 12, 14, 16, 19]);
      box.add(data);
      data = new twaver.Element();
      data.setStyle("chart.values", [21, 23, 25, 23, 28]);
      box.add(data);

      chart.setXScaleTexts(new twaver.List(['A', 'B', 'C', 'D', 'E']));
      chart.setLowerLimit(0);
      chart.setYScaleValueGap(5);
      chart.setXAxisText('A Line Chart');
      chart.setBackgroundVisible(true);

      var chartDom = chart.getView();
      chartDom.style.width = "200px";
      chartDom.style.height = "150px";
      document.body.appendChild(chartDom);
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

运行结果



PieChart

TWaver HTML5 饼图也是与dataBox数据容器相关联，其上的每一个扇面对应一个数据元素，扇面的大小由这个数据元素的chart value值的大小决定。

饼图种类

饼图具有环状，椭圆，矩形等多种表现形式

```
chart.setType("ovalDonut");
```

饼图类型：

- oval - 椭圆
- circle - 圆形
- line - 条形
- donut - 挖空
- ovalDonut - 椭圆挖空

```
createData('A', 10, "#4CB743");
createData('B', 20, "#D6EAC9");
createData('C', 40, "#77DD77");

chart.setType("ovalDonut");
chart.formatValueText = function(value, data){
    return data.getName() + ' - ' + parseInt(value/chart.getSum()*100) + '%';
};

var chartDom = chart.getView();
chartDom.style.width = "200px";
chartDom.style.height = "150px";
document.body.appendChild(chartDom);
}
function createData(name, value, color){
    var data = new twaver.Element();
    data.setStyle("chart.value", value);
    data.setStyle("chart.color", color);
    data.setStyle("chart.value.color", 0x555555);
    data.setStyle("chart.value.font", 'hevetica');
    data.setName(name);
    box.add(data);
};
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>
```

运行界面



DialChart

仪表盘，使用指针表示数值，通常每个指针对应一个数据元素。仪表盘主要由三部分组成：表盘，轴，指针，这三部分都可以设置不同的颜色和渲染方式。

示例：

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo </title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var box;
    function init() {
      var chart = new twaver.charts.DialChart();
      box = chart.getDataBox();

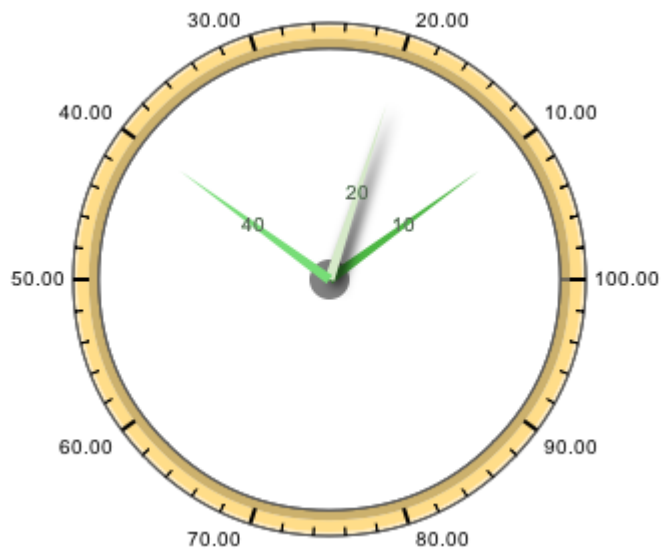
      createData('A', 10, "#4CB743");
      createData('B', 20, "#D6EAC9");
      createData('C', 40, "#77DD77");

      chart.setOutlineColor("#555555");
      chart.setOutlineWidth(1);
      chart.setOuterBrighterRadius(2);
      chart.setInnerDarkerRadius(5);
      chart.setColorRangeFillColor("#FFDD88");
      chart.setInnerRadius(0.9);
      chart.setScaleInside(false);
      chart.setScaleLowerLimitTextVisible(false);

      var chartDom = chart.getView();
      chartDom.style.width = "400px";
      chartDom.style.height = "300px";
      document.body.appendChild(chartDom);
    }

    function createData(name, value, color){
      var data = new twaver.Element();
      data.setStyle("chart.value", value);
      data.setStyle("chart.color", color);
      data.setStyle("chart.value.color", "#555555");
      data.setStyle("chart.value.font", 'hevetica');
      data.setStyle('dialchart.radius', 0.8);
      data.setName(name);
      box.add(data);
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

运行结果：



BubbleChart

气泡图，使用三维数值（x, y, value）表现元素的特性，每一组值对应一个气泡，x, y 表示气泡的位置，value值表示气泡的大小，通常一个数据元素对应一个气泡，如下代码定义一个气泡：

```
var data = new twaver.Element();
data.setStyle("chart.values", [80]);
data.setStyle("chart.xaxis.values", [5]);
data.setStyle("chart.yaxis.values", [5]);
data.setStyle("chart.color", "#BD6F17");
box.add(data);
```

也可以让一个元素对应多个气泡，如下设置

```
data = new twaver.Element();
data.setStyle("chart.values", [50, 70]);
data.setStyle("chart.xaxis.values", [20,22]);
data.setStyle("chart.yaxis.values", [20,12]);
data.setStyle("chart.color", "#651600");
box.add(data);
```

示例：

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var box;
    function init() {
      var chart = new twaver.charts.BubbleChart();
      box = chart.getDataBox();

      var data = new twaver.Element();
      data.setStyle("chart.values", [80]);
      data.setStyle("chart.xaxis.values", [5]);
      data.setStyle("chart.yaxis.values", [5]);
      data.setStyle("chart.color", "#BD6F17");
      box.add(data);
      data = new twaver.Element();
      data.setStyle("chart.values", [100]);
      data.setStyle("chart.xaxis.values", [10]);
      data.setStyle("chart.yaxis.values", [30]);
      data.setStyle("chart.color", "#883000");
      box.add(data);
      data = new twaver.Element();
      data.setStyle("chart.values", [50, 70]);
      data.setStyle("chart.xaxis.values", [20,22]);
      data.setStyle("chart.yaxis.values", [20,12]);
      data.setStyle("chart.color", "#651600");
      box.add(data);

      chart.setXScaleTexts(new twaver.List(["1", "2", "3", "4", "5", "6", "7", "8"]));

      var chartDom = chart.getView();
```

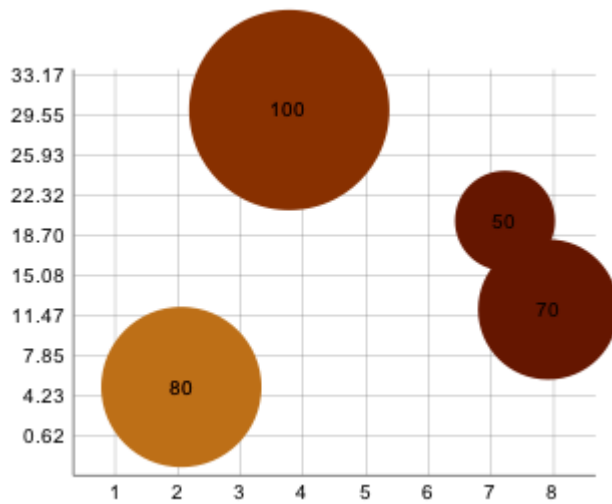


```

    chartDom.style.width = "400px";
    chartDom.style.height = "300px";
    document.body.appendChild(chartDom);
  }
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```

运行界面：



RadarChart

雷达图，用于表现多维数据的图表形式。

每个数据元素可以设置一组 [chart values](#)，用以表示多维度的值,如下元素设置了五维数值

```
data.setStyle("chart.values", [80, 100, 150, 230, 40]);
```

示例：

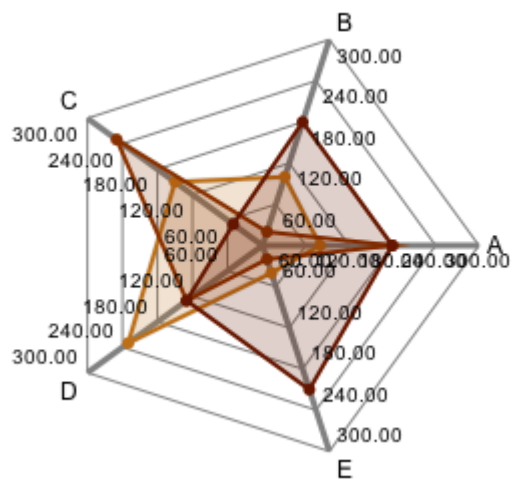
```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo </title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    var box;
    function init() {
      var chart = new twaver.charts.RadarChart();
      box = chart.getDataBox();

      var data = new twaver.Element();
      data.setStyle("chart.values", [80, 100, 150, 230, 40]);
      data.setStyle("chart.color", "#BD6F17");
      box.add(data);
      data = new twaver.Element();
      data.setStyle("chart.values", [180, 20, 250, 130, 20]);
      data.setStyle("chart.color", "#883000");
      box.add(data);
      data = new twaver.Element();
      data.setStyle("chart.values", [180, 180, 50, 130, 210]);
      data.setStyle("chart.color", "#651600");
      box.add(data);

      chart.setAxisList(new twaver.List([
        {text:"A", min:0, max:300},
        {text:"B", min:0, max:300},
        {text:"C", min:0, max:300},
        {text:"D", min:0, max:300},
        {text:"E", min:0, max:300}
      ]));

      var chartDom = chart.getView();
      chartDom.style.width = "300px";
      chartDom.style.height = "300px";
      document.body.appendChild(chartDom);
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>
```

运行界面



告警的使用

前面[告警元素](#)章节简单介绍了告警的使用，本章将详细介绍TWaver中告警模型（告警，告警级别，告警状态，告警容器），告警机制和告警的呈现。

告警 - twaver.Alarm，表示告警数据的对象，包含告警级别，告警发生时间，与其关联的网元编号等属性，在[告警元素](#)章节有介绍

告警级别 - twaver.AlarmSeverity，表示告警的紧急程度，TWaver默认提供六级告警级别（CRITICAL, MAJOR, MINOR, WARNING, INDETERMINATE, CLEARED），其中value属性反映告警的严重程度

告警容器 - twaver.AlarmBox，用于管理告警的数据容器，提供告警的增删改查操作，详见[数据容器](#)

网元的告警状态 - twaver.AlarmState，反映网元当前的告警状态，如该设备是否有告警发生，是否有传递告警，总共有多少告警，最高告警级别是什么等信息，AlarmState 只包含告警的状态信息，并不包含具体的告警数据，所以AlarmState上无法获得某个具体的Alarm，更无法获得某个告警的具体属性，就像一个州的人口统计信息报表，不包涵具体某个人的信息

告警统计 - twaver.AlarmStateStatistics，上面说的网元告警状态实际就是单个网元上的告警统计信息，扩展到整个网元容器，要查看整个ElementBox或者指定网元的告警统计信息，可以用AlarmStateStatistics

- [告警级别](#)
- [告警状态与统计](#)
- [告警呈现](#)

告警级别

告警级别：用于描述告警紧急程度的对象，TWaver HTML5中使用twaver.AlarmSeverity类定义告警级别，包含name，nickName，value，color等属性

默认告警级别

TWaver HTML5中预定义了六种告警级别，告警级别的value属性表示告警的严重程度，默认值越大告警越严重

```
var AlarmSeverity = twaver.AlarmSeverity;
AlarmSeverity.CRITICAL = AlarmSeverity.add(500, "Critical", "C", '#FF0000');
AlarmSeverity.MAJOR = AlarmSeverity.add(400, "Major", "M", '#FFA000');
AlarmSeverity.MINOR = AlarmSeverity.add(300, "Minor", "m", '#FFFF00');
AlarmSeverity.WARNING = AlarmSeverity.add(200, "Warning", "W", '#00FFFF');
AlarmSeverity.INDETERMINATE = AlarmSeverity.add(100, "Indeterminate", "N", '#C800FF');
AlarmSeverity.CLEARED = AlarmSeverity.add(0, "Cleared", "R", '#00FF00');
```

Severity	Letter	Value	Color
CRITICAL	C	500	Red
MAJOR	M	400	Orange
MINOR	m	300	Yellow
WARNING	W	200	Cyan
INDETERMINATE	N	100	Purple
CLEARED	R	0	Green

扩展告警级别

告警级别中的级别都是静态变量，用户也可以全局注册或者卸载自己的告警级别，此外还提供清除所有告警级别的方法（因为是全局变量，删除告警级别会对整个程序影响，要小心使用）

```
twaver.AlarmSeverity.add = function (value, name, nickName, color, displayName)
twaver.AlarmSeverity.remove = function (name)
twaver.AlarmSeverity.clear = function ()
```

如下的代码可以实现清除默认告警级别，注册自己的告警级别

```
□twaver.AlarmSeverity.clear();
□twaver.AlarmSeverity.add(1, "a", "a", "#FF0000", "AAA");□
```

扩展告警级别示例

下面的例子清除了默认的告警级别，注册了几种定制的告警级别，并在拓扑图中使用

```

<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var network = new twaver.network.Network();

      var box = network.getElementBox();

      twaver.AlarmSeverity.clear();
      var a = twaver.AlarmSeverity.add(1, "a", "a", "#FF0000", "AAA");
      var b = twaver.AlarmSeverity.add(2, "b", "b", "#0000FF", "BBB");
      var c = twaver.AlarmSeverity.add(3, "c", "c", "#00FF00", "CCC");

      var node1 = new twaver.Group();
      node1.setExpanded(true);
      var node2 = new twaver.Node();
      node2.setLocation(100, 100);
      var node3 = new twaver.Node();
      node3.setLocation(200, 150);
      node3.setParent(node1);
      node2.setParent(node1);
      box.add(node1);
      box.add(node2);
      box.add(node3);

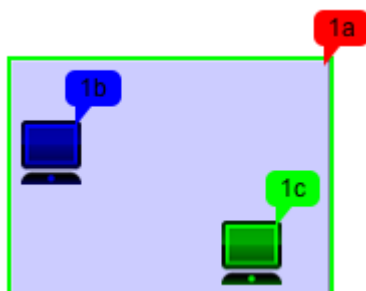
      addAlarm(box.getAlarmBox(), node1, a);
      addAlarm(box.getAlarmBox(), node2, b);
      addAlarm(box.getAlarmBox(), node3, c);

      var networkDom = network.getView();
      networkDom.style.width = "100%";
      networkDom.style.height = "100%";
      document.body.appendChild(networkDom);
    }

    function addAlarm(alarmBox, element, severity){
      var alarm = new twaver.Alarm(null, element.getId(), severity);
      alarmBox.add(alarm);
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
</html>

```

运行结果



告警状态与统计

告警状态

告警状态用于描述网元的告警统计状态，是否有告警，最高级别告警是什么，有多少新发告警，有多少传递告警.....告警状态并不包含具体的告警，TWaver HTML5中用twaver.AlarmState类表示网元的告警状态，我们可以通过element.getAlarmState()获取该对象。

```
var alarmState = node.getAlarmState();
```

twaver.AlarmState - 告警状态

告警状态属性包括，确认告警的最高级别，新发告警的最高级别，该网元所有告警的最高级别，新发告警次高级别，自身告警最高级别，传递告警级别以及各级别告警的数量

```
getHighestAcknowledgedAlarmSeverity: function ()
getHighestNewAlarmSeverity: function ()
getHighestOverallAlarmSeverity: function ()
getHighestNativeAlarmSeverity: function ()
hasLessSevereNewAlarms: function ()
```

修改告警状态的相关方法：确认告警，清除告警，增加/减少确认告警，删除告警...

```
increaseAcknowledgedAlarm: function (severity, increment)
increaseNewAlarm: function (severity, increment)
decreaseAcknowledgedAlarm: function (severity, decrement)
decreaseNewAlarm: function (severity, decrement)
acknowledgeAlarm: function (severity)
acknowledgeAllAlarms: function (severity)
getAcknowledgedAlarmCount: function (severity)
getAlarmCount: function (severity)
getNewAlarmCount: function (severity)
setNewAlarmCount: function (severity, count)
removeAllNewAlarms: function (severity)
setAcknowledgedAlarmCount: function (severity, count)
removeAllAcknowledgedAlarms: function (severity)
isEmpty: function ()
clear: function ()
```

其他方法

```
getPropagateSeverity: function ()
setPropagateSeverity: function (propagateSeverity)
isEnabledPropagation: function ()
setEnabledPropagation: function (enablePropagation)
```

告警统计

告警状态统计：对整个elementBox中所有网元告警状态进行统计，在TWaver HTML5中我们用twaver.AlarmStateStatistics实现告警统计功能，使用告警统计可以快速的获取各种级别告警、新发告警、确认告警的数量，并支持过滤器，可指定对某些网元做统计

AlarmStateStatistics - 告警统计器

创建一个告警统计器

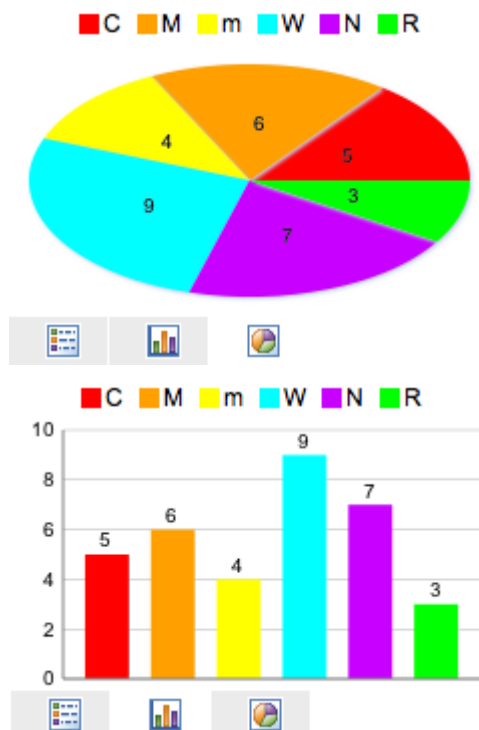
```
var statistics= new twaver.AlarmStateStatistics(elementBox);
```

统计器提供的方法函数




```
//获取新发告警数量
getNewAlarmCount: function (severity)
//获取确认告警数量
getAcknowledgedAlarmCount: function (severity)
//获取告警总数量
getTotalAlarmCount: function (severity)
//设置过滤器
setFilterFunction: function (f)
```

告警统计的呈现

将告警统计的信息放置到表格中，呈现下面的效果



Severity	New	Acked	Total
MAJOR	5	1	6
WARNING	8	1	9
CLEARED	2	1	3
CRITICAL	3	2	5
MINOR	1	3	4
INDETERMINATE	0	7	7
TOTAL	19	15	34

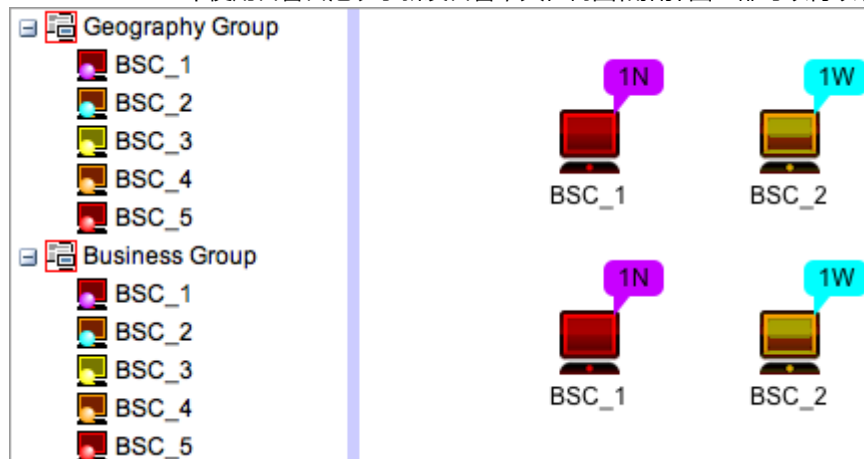




告警呈现

告警有多种呈现方式，不同类型的告警在不同的组件上有不同的表现效果

告警冒泡

TWaver HTML5中使用告警冒泡表示新发告警，其在树图和拓扑图上都可以得以表现



告警渲染（染色，边框）

上图中网元图片的颜色渲染，树节点图标渲染以及边框都属于告警渲染的效果

告警表格

告警最传统的表现方式是放在表格中列举，使用TWaver的表格组件关联上告警容器，就可以在表格中显示告警信息，其每一行表示一条告警

Mapping ID	Alarm Severity	Acked	Cleared	Raised Time
1	Indeterminate	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
2	Warning	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
3	Minor	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
4	Major	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
5	Critical	<input type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
5	Indeterminate	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
4	Warning	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54
3	Minor	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2011-11-22 15:29:54

告警统计图表

告警统计信息通常使用图表的形式表现

Severity	New	Acked	Total
MAJOR	5	1	6
WARNING	8	1	9
CLEARED	2	1	3
CRITICAL	3	2	5
MINOR	1	3	4
INDETERMINATE	0	7	7
TOTAL	19	15	34



告警表格示例

下面的例子创建了十条随机告警，并在告警表格中显示，其中告警列借用了TWaver HTML5 Demo中的相关代码

```
<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init(){
      var box = new twaver.ElementBox();
      var alarmBox = box.getAlarmBox();
      var table = new twaver.controls.Table(alarmBox);

      var node = new twaver.Node();
      box.add(node);

      var i = 0;
      while(i++ < 10){
        var alarm = new twaver.Alarm(i, node.getId(), randomSeverity());
        alarm.setAked(Math.random() >= 0.5);
        alarm.setCleared(Math.random() >= 0.5);
        alarmBox.add(alarm);
      }

      createColumn(table,'Id', 'id', 'accessor', 'string').setWidth(30);
      createColumn(table,'Alarm Severity', 'alarmSeverity', 'accessor').setHorizontalAlign('center');
      createColumn(table,'Aked', 'acked', 'accessor', "boolean");
      createColumn(table,'Cleared', 'cleared', 'accessor', "boolean");

      table.onCellRendered = function (params) {
        if (params.column.getName() === 'Alarm Severity') {
          params.div.style.backgroundColor = params.data.getAlarmSeverity().color;
        }
      };

      var tablePane = new twaver.controls.TablePane(table);
      var tableDom = tablePane.getView();
      tableDom.style.width = "100%";
      tableDom.style.height = "100%";
      document.body.appendChild(tableDom);
      window.onresize = function(){
        tablePane.invalidate();
      }
    }

    function randomSeverity() {
      var severities = twaver.AlarmSeverity.severities;
      return severities.get(Math.floor(Math.random() * severities.size()));
    }

    function createColumn(table, name, propertyName, propertyType, valueType) {
      var column = new twaver.Column(name);
      column.setName(name);
      column.setPropertyName(propertyName);
      column.setPropertyType(propertyType);
      if (valueType) column.setValueType(valueType);
      table.getColumnBox().add(column);
      return column;
    }
  </script>
</head>
<body onload="init()" style="margin:0;"></body>
```

```
</html>
```

运行界面

Id	Alarm Severity	Acked	Cleared
1	Warning	<input type="checkbox"/>	<input type="checkbox"/>
2	Major	<input type="checkbox"/>	<input type="checkbox"/>
3	Warning	<input type="checkbox"/>	<input type="checkbox"/>
4	Cleared	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	Major	<input type="checkbox"/>	<input type="checkbox"/>
6	Minor	<input type="checkbox"/>	<input type="checkbox"/>
7	Warning	<input type="checkbox"/>	<input type="checkbox"/>
8	Warning	<input checked="" type="checkbox"/>	<input type="checkbox"/>

附录

- [网元样式表](#)

网元样式表

TWaver HTML5网元的所有图形属性都是通过设置样式属性来实现，本章将枚举各种网元类型的样式名，赋值范围和使用说明。

TWaver HTML5中的样式设置如下：

```
var link = new Link(from, to);
link.setName("hello TWaver!");
//样式名：link.type
//表示设置连线类型为flexional
link.setStyle("link.type", "flexional");
```

alarm - 告警冒泡相关样式

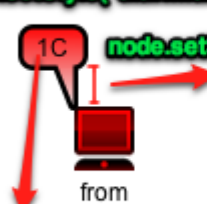
样式名	赋值范围	说明
alarm.alpha	Number，默认值为：1	告警冒泡透明度
alarm.cap	String，默认值为：butt 取值范围：butt, round, square	边框线端点样式
alarm.color	Color，默认值为：#000000	填充色
alarm.corner.radius	Number，默认值为：5	圆角半径
alarm.direction	String，默认值为：aboveright 取值范围：aboveleft, aboveright, belowleft, belowright, leftabove, leftbelow, rightabove, rightbelow, above, below, left, right	冒泡方向
alarm.gradient	String，默认值为：none 取值范围：linear.east, linear.north, linear.northeast, linear.northwest, linear.south, linear.southeast, linear.southwest, linear.west, none, radial.center, radial.east, radial.north, radial.northeast, radial.northwest, radial.south, radial.southeast, radial.southwest, radial.west, spread.antidiagonal, spread.diagonal, spread.east, spread.horizontal, spread.north, spread.south, spread.vertical, spread.west	渐变填充样式
alarm.gradient.color	Color，默认值为：#FFFFFF	渐变填充色
alarm.join	String，默认值为：miter 取值范围：miter, round, bevel	边框线拐点样式
alarm.outline.color	Color，默认值为：#000000	边框线颜色
alarm.outline.width	Number，默认值为：-1	边框线宽度
alarm.padding	Number，默认值为：0	间隙宽度

alarm.padding.bottom	Number , 默认值为 : 0	底部间隙
alarm.padding.left	Number , 默认值为 : 0	左边间隙
alarm.padding.right	Number , 默认值为 : 0	右边间隙
alarm.padding.top	Number , 默认值为 : 0	顶部间隙
alarm.pointer.length	Number , 默认值为 : 10	指针长度
alarm.pointer.width	Number , 默认值为 : 8	指针宽度
alarm.position	String , 默认值为 : hotspot 取值范围 : topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright, topleft.bottomleft, topleft.bottomright, top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft, bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	冒泡位置
alarm.shadowable	String , 默认值为 : true	冒泡阴影
alarm.xoffset	Number , 默认值为 : 0	x偏移量
alarm.yoffset	Number , 默认值为 : 0	y偏移量

```

node.setStyle("alarm.position", "topleft.topleft");
node.setStyle("alarm.direction", "aboveleft");
node.setStyle("alarm.pointer.length", 20);
node.setStyle("alarm.gradient", "linear.south");
node.setStyle("alarm.gradient.color", "#EEEEEE");

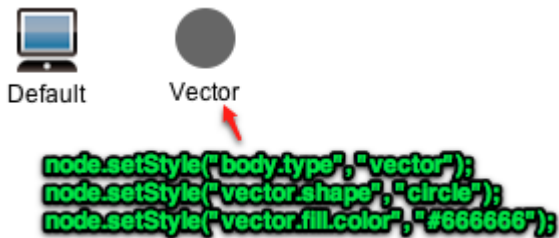
```



body - 网元主体样式

样式名	赋值范围	说明
body.type	String , 默认值为 : default	网元主体类型

	取值范围：none, 'default', 'vector', 'default.vector', 'vector.default'	
--	--	--



bus - 总线样式

样式名	赋值范围	说明
bus.style	String，默认值为：nearby 取值范围：'nearby', 'north', 'south', 'west', 'east'	总线类型

示例：

```
var node = new twaver.Node();
node.setSize(16, 16);
node.setLocation(200, 300);
box.add(node);

var node2 = new twaver.Node();
node2.setSize(16, 16);
node2.setLocation(100, 400);
box.add(node2);

var bus = new twaver.Bus();
bus.addPoint({x: 50, y: 350});
bus.addPoint({x: 200, y: 350});
bus.addPoint({x: 200, y: 420});
bus.addPoint({x: 50, y: 420});
bus.setStyle("vector.fill.color", "#00ff00");
bus.setStyle('vector.outline.width', 5);
bus.setStyle('bus.style', 'north');
box.add(bus);

box.add(new twaver.Link(node, bus));
box.add(new twaver.Link(node2, bus));
```

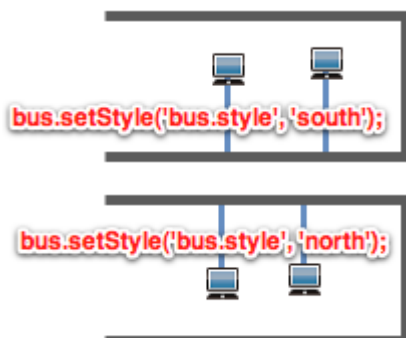
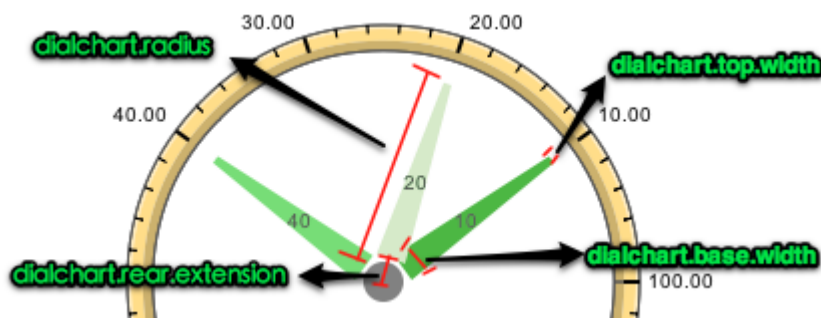


chart - 图表样式

样式名	赋值范围	说明
chart.bubble.shape	String , 默认值为 : circle 取值范围 : rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	气泡形状
chart.line.width	Number , 默认值为 : 2	线宽度
chart.marker.shape	String , 默认值为 : circle 取值范围 : rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	标记形状
chart.marker.size	Number , 默认值为 : 6	标记尺寸
chart.value	Number , 默认值为 : 0	图表数值
chart.value.color	Color , 默认值为 : #000000	图表颜色

dialchart - 仪表盘样式

样式名	赋值范围	说明
dialchart.radius	Number , 默认值为 : 0.8	指针长度比例
dialchart.base.width	Number , 默认值为 : 5	指针宽度
dialchart.rear.extension	Number , 默认值为 : 0	指针偏移量
dialchart.top.width	Number , 默认值为 : 0	指针尖宽度



follower - 跟随者相关样式

样式名	赋值范围	说明
follower.column.index	Number , 默认值为 : 0	列号
follower.column.span	Number , 默认值为 : 1	跨列数
follower.padding	Number , 默认值为 : 0	间隙

follower.padding.bottom	Number , 默认值为 : 0	底部间隙
follower.padding.left	Number , 默认值为 : 0	左边间隙
follower.padding.right	Number , 默认值为 : 0	右边间隙
follower.padding.top	Number , 默认值为 : 0	顶部间隙
follower.row.index	Number , 默认值为 : 0	行号
follower.row.span	Number , 默认值为 : 1	跨行数

grid - 网格样式

样式名	赋值范围	说明
grid.border	Number , 默认值为 : 1	边距宽度
grid.border.bottom	Number , 默认值为 : 0	底部边距
grid.border.left	Number , 默认值为 : 0	左边边距
grid.border.right	Number , 默认值为 : 0	右边边距
grid.border.top	Number , 默认值为 : 0	顶部边距
grid.cell.deep	Number , 默认值为 : -1	单元格深度
grid.column.count	Number , 默认值为 : 1	列数
grid.deep	Number , 默认值为 : 1	网格深度
grid.fill	String , 默认值为 : true	是否填充
grid.fill.color	Color , 默认值为 : #C0C0C0	填充色
grid.padding	Number , 默认值为 : 1	间隙
grid.padding.bottom	Number , 默认值为 : 0	底部间隙
grid.padding.left	Number , 默认值为 : 0	左边间隙
grid.padding.right	Number , 默认值为 : 0	右边间隙
grid.padding.top	Number , 默认值为 : 0	顶部间隙
grid.row.count	Number , 默认值为 : 1	行数

示例 :

```

<!DOCTYPE html>
<html>
<head>
  <title>TWaver HTML5 Demo</title>
  <script type="text/javascript" src="../demo/twaver.js"></script>
  <script type="text/javascript">
    function init() {
      var network = new twaver.network.Network();
      var box = network.getElementBox();
    }
  </script>
</head>
</html>

```

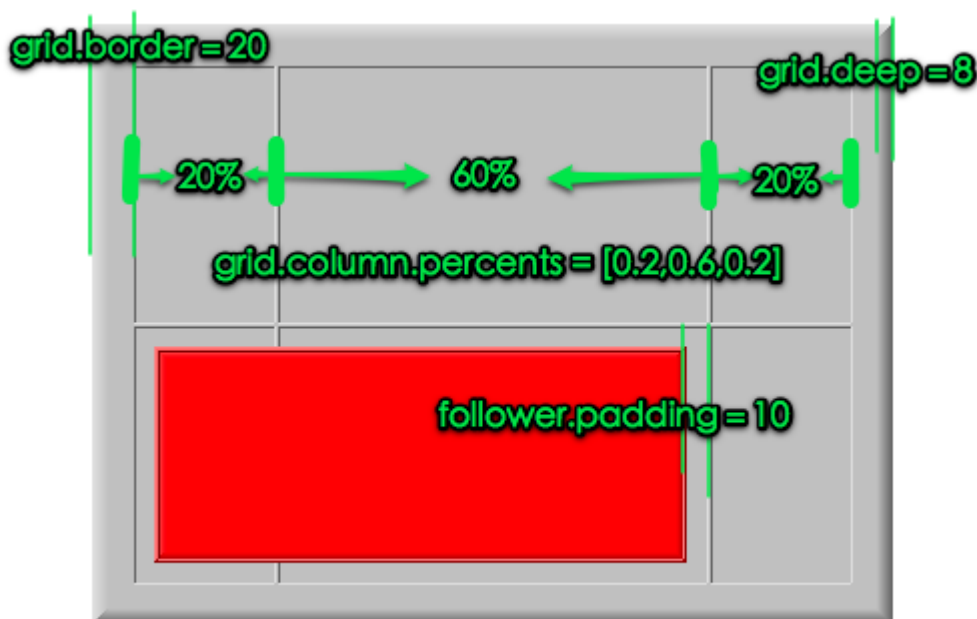
```

var grid = new twaver.Grid();
grid.setLocation(20, 20);
grid.setSize(400, 300);
grid.setStyle("grid.border", 20);
grid.setStyle("grid.row.count", 2);
grid.setStyle("grid.column.count", 3);
grid.setStyle("grid.column.percents", [0.2,0.6,0.2]);
grid.setStyle("grid.deep", 8);
box.add(grid);

var cell=new twaver.Grid();
cell.setStyle("follower.column.index",0);
cell.setStyle("follower.row.index",1);
cell.setStyle("follower.column.span",2);
cell.setStyle("grid.fill.color", "#ff0000");
cell.setStyle("follower.padding",10);
cell.setHost(grid);
grid.addChild(cell);
box.add(cell);

var networkDom = network.getView();
networkDom.style.width = "100%";
networkDom.style.height = "100%";
document.body.appendChild(networkDom);
}
</script>
</head>
<body onload="init()" style="margin:0;"> </body>
</html>

```



group - 分组相关样式

样式名	赋值范围	说明
group.cap	String , 默认值为 : butt	边框线端点样式

	取值范围：butt, round, square	
group.deep	Number，默认值为：1	分组深度
group.fill	String，默认值为：true	是否填充
group.fill.color	Color，默认值为：#CCCCFF	填充色
group.gradient	String，默认值为：none 取值范围：linear.east, linear.north, linear.northeast, linear.northwest, linear.south, linear.southeast, linear.southwest, linear.west, none, radial.center, radial.east, radial.north, radial.northeast, radial.northwest, radial.south, radial.southeast, radial.southwest, radial.west, spread.antidiagonal, spread.diagonal, spread.east, spread.horizontal, spread.north, spread.south, spread.vertical, spread.west	渐变填充
group.gradient.color	Color，默认值为：#FFFFFF	渐变色
group.join	String，默认值为：miter 取值范围：miter, round, bevel	边框线连结点样式
group.outline.color	Color，默认值为：#5B5B5B	边框线颜色
group.outline.width	Number，默认值为：-1	边框线宽度
group.padding	Number，默认值为：5	间隙
group.padding.bottom	Number，默认值为：0	底部间隙
group.padding.left	Number，默认值为：0	左边间隙
group.padding.right	Number，默认值为：0	右边间隙
group.padding.top	Number，默认值为：0	顶部间隙
group.shape	String，默认值为：rectangle 取值范围：rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	分组形状

icons - 图标样式

样式名	赋值范围	说明
icons.orientation	String，默认值为：right 取值范围：left, right, top, bottom	图标排列方向
icons.position	String，默认值为： topleft.bottomright 取值范围：topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright, topleft.bottomleft, topleft.bottomright,	图标位置

	top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft, bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	
icons.xgap	Number , 默认值为 : 1	x间隙
icons.xoffset	Number , 默认值为 : 0	x偏移量
icons.ygap	Number , 默认值为 : 1	y间隙
icons.yoffset	Number , 默认值为 : 0	y偏移量

image - 图片样式

样式名	赋值范围	说明
image.padding	Number , 默认值为 : 0	间隙
image.padding.bottom	Number , 默认值为 : 0	底部间隙
image.padding.left	Number , 默认值为 : 0	左边间隙
image.padding.right	Number , 默认值为 : 0	右边间隙
image.padding.top	Number , 默认值为 : 0	顶部间隙

label - 标签样式

样式名	赋值范围	说明
label.alpha	Number , 默认值为 : 1	透明度
label.cap	String , 默认值为 : butt 取值范围 : butt, round, square	边框线端点样式
label.color	Color , 默认值为 : #000000	文本颜色
label.corner.radius	Number , 默认值为 : 0	边框圆角半径
label.direction	String , 默认值为 : below 取值范围 : aboveleft, aboveright, belowleft, belowright, leftabove, leftbelow, rightabove, rightbelow, above, below, left, right	标签方位, 必须设置指针长度时才有效, 见: label.pointer.length
label.fill	String , 默认值为 : false	背景填充
label.fill.color	Color , 默认值为 : #C0C0C0	背景填充色

label.gradient	String , 默认值为 : none 取值范围 : linear.east, linear.north, linear.northeast, linear.northwest, linear.south, linear.southeast, linear.southwest, linear.west, none, radial.center, radial.east, radial.north, radial.northeast, radial.northwest, radial.south, radial.southeast, radial.southwest, radial.west, spread.antidiagonal, spread.diagonal, spread.east, spread.horizontal, spread.north, spread.south, spread.vertical, spread.west	背景渐变
label.gradient.color	Color , 默认值为 : #FFFFFF	背景渐变色
label.join	String , 默认值为 : miter 取值范围 : miter, round, bevel	边框线拐点连接样式
label.outline.color	Color , 默认值为 : #000000	边框线颜色
label.outline.width	Number , 默认值为 : -1	边框线宽度
label.padding	Number , 默认值为 : 0	间隙
label.padding.bottom	Number , 默认值为 : 0	底部间隙
label.padding.left	Number , 默认值为 : 0	左边间隙
label.padding.right	Number , 默认值为 : 0	右边间隙
label.padding.top	Number , 默认值为 : 0	顶部间隙
label.pointer.length	Number , 默认值为 : 0	指针长度
label.pointer.width	Number , 默认值为 : 8	指针宽度
label.position	String , 默认值为 : bottom.bottom 取值范围 : topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright, topleft.bottomleft, topleft.bottomright, top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft, bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	位置
label.shadowable	String , 默认值为 : true	阴影
label.xoffset	Number , 默认值为 : 0	x偏移量

label.yoffset	Number , 默认值为 : 2	y偏移量
---------------	-------------------	------

link - 连线样式

样式名	赋值范围	说明
link.bundle.enable	String , 默认值为 : true	是否参与捆绑
link.bundle.expanded	String , 默认值为 : true	展开
link.bundle.gap	Number , 默认值为 : 12	捆绑间隔
link.bundle.id	Number , 默认值为 : 0	捆绑编号
link.bundle.independent	String , 默认值为 : false	独立捆绑
link.bundle.offset	Number , 默认值为 : 20	捆绑偏移量
link.cap	String , 默认值为 : butt 取值范围 : butt, round, square	连线端点样式
link.color	Color , 默认值为 : #658DC1	颜色
link.corner	String , 默认值为 : round 取值范围 : none, round, bevel	圆角类型
link.extend	Number , 默认值为 : 20	延伸量
link.from.at.edge	String , 默认值为 : true	连接到起始节点的边缘
link.from.position	String , 默认值为 : center 取值范围 : topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright, topleft.bottomleft, topleft.bottomright, top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft, bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	起始连结点位置
link.from.xoffset	Number , 默认值为 : 0	起始连结点x偏移量
link.from.yoffset	Number , 默认值为 : 0	起始连结点y偏移量
link.handler.alpha	Number , 默认值为 : 1	连线手柄透明度
link.handler.cap	String , 默认值为 : butt 取值范围 : butt, round, square	连线手柄边框线端点样式
link.handler.color	Color , 默认值为 : #000000	连线手柄颜色

link.handler.corner.radius	Number , 默认值为 : 0	连线手柄圆角半径
link.handler.direction	String , 默认值为 : below 取值范围 : aboveleft, aboveright, belowleft, belowright, leftabove, leftbelow, rightabove, rightbelow, above, below, left, right	连线手柄方向
link.handler.fill	String , 默认值为 : false	连线手柄是否填充
link.handler.fill.color	Color , 默认值为 : #C0C0C0	连线手柄填充色
link.handler.gradient	String , 默认值为 : none 取值范围 : linear.east, linear.north, linear.northeast, linear.northwest, linear.south, linear.southeast, linear.southwest, linear.west, none, radial.center, radial.east, radial.north, radial.northeast, radial.northwest, radial.south, radial.southeast, radial.southwest, radial.west, spread.antidiagonal, spread.diagonal, spread.east, spread.horizontal, spread.north, spread.south, spread.vertical, spread.west	连线手柄填充渐变
link.handler.gradient.color	Color , 默认值为 : #FFFFFF	连线手柄渐变颜色
link.handler.join	String , 默认值为 : miter 取值范围 : miter, round, bevel	连线手柄边框线拐点样式
link.handler.outline.color	Color , 默认值为 : #000000	连线手柄边框线颜色
link.handler.outline.width	Number , 默认值为 : -1	连线手柄边框宽度
link.handler.padding	Number , 默认值为 : 0	连线手柄间隙
link.handler.padding.bottom	Number , 默认值为 : 0	连线手柄底部间隙
link.handler.padding.left	Number , 默认值为 : 0	连线手柄左边间隙
link.handler.padding.right	Number , 默认值为 : 0	连线手柄右边间隙
link.handler.padding.top	Number , 默认值为 : 0	连线手柄顶部间隙
link.handler.pointer.length	Number , 默认值为 : 0	连线手柄指针长度
link.handler.pointer.width	Number , 默认值为 : 8	连线手柄指针宽度
link.handler.position	String , 默认值为 : topleft.topleft 取值范围 : topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright, topleft.bottomleft, topleft.bottomright, top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft,	连线手柄位置

	bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	
link.handler.shadowable	String , 默认值为 : true	连线手柄是否阴影
link.handler.xoffset	Number , 默认值为 : 0	连线手柄x偏移量
link.handler.yoffset	Number , 默认值为 : 0	连线手柄y偏移量
link.join	String , 默认值为 : miter 取值范围 : miter, round, bevel	连线拐点样式
link.looped.direction	String , 默认值为 : northwest 取值范围 : north, northeast, east, southeast, south, southwest, west, northwest	自环方向
link.looped.gap	Number , 默认值为 : 6	自环间隙
link.looped.type	String , 默认值为 : arc	自环类型
link.split.by.percent	String , 默认值为 : true	按百分比劈分
link.split.percent	Number , 默认值为 : 0.5	劈分百分比
link.split.value	Number , 默认值为 : 20	劈分量
link.to.at.edge	String , 默认值为 : true	连接到结束节点边缘
link.to.position	String , 默认值为 : center 取值范围 : topleft.topleft, topleft.topright, top.top, topright.topleft, topright.topright, topleft, top, topright, topleft.bottomleft, topleft.bottomright, top.bottom, topright.bottomleft, topright.bottomright, left.left, left, left.right, center, right.left, right, right.right, bottomleft.topleft, bottomleft.topright, bottom.top, bottomright.topleft, bottomright.topright, bottomleft, bottom, bottomright, bottomleft.bottomleft, bottomleft.bottomright, bottom.bottom, bottomright.bottomleft, bottomright.bottomright	结束端连结点位置
link.to.xoffset	Number , 默认值为 : 0	结束端连结点x偏移量
link.to.yoffset	Number , 默认值为 : 0	结束端连结点y偏移量
link.type	String , 默认值为 : arc 取值范围 : arc, triangle, parallel, flexional, flexional.horizontal, flexional.vertical, orthogonal, orthogonal.horizontal,	连线类型

	orthogonal.vertical, orthogonal.H.V, orthogonal.V.H, extend.top, extend.left, extend.bottom, extend.right	
link.width	Number , 默认值为 : 3	连线宽度
link.xradius	Number , 默认值为 : 8	x圆角半径
link.yradius	Number , 默认值为 : 8	y圆角半径

outer - 边框

样式名	赋值范围	说明
outer.cap	String , 默认值为 : butt 取值范围 : butt, round, square	边框线端点样式
outer.join	String , 默认值为 : miter 取值范围 : miter, round, bevel	边框线拐点样式
outer.padding	Number , 默认值为 : 1	边框间隙
outer.padding.bottom	Number , 默认值为 : 0	边框底部间隙
outer.padding.left	Number , 默认值为 : 0	边框左边间隙
outer.padding.right	Number , 默认值为 : 0	边框右边间隙
outer.padding.top	Number , 默认值为 : 0	边框顶部间隙
outer.shape	String , 默认值为 : rectangle 取值范围 : rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	边框形状
outer.width	Number , 默认值为 : 2	边框线宽度

select - 选中边框样式

样式名	赋值范围	说明
select.cap	String , 默认值为 : butt 取值范围 : butt, round, square	边框线端点样式
select.color	String , 默认值为 : rgba(0, 0, 0, 0.7)	颜色
select.join	String , 默认值为 : miter 取值范围 : miter, round, bevel	边框线拐点样式
select.padding	Number , 默认值为 : 2	间隙
select.padding.bottom	Number , 默认值为 : 0	底部间隙
select.padding.left	Number , 默认值为 : 0	左边间隙
select.padding.right	Number , 默认值为 : 0	右边间隙
select.padding.top	Number , 默认值为 : 0	顶部间隙

select.shape	String , 默认值为 : rectangle 取值范围 : rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	形状
select.style	String , 默认值为 : shadow 取值范围 : null, shadow, border	选中边框样式
select.width	Number , 默认值为 : 2	选中边框宽度

shadow - 阴影

样式名	赋值范围	说明
shadow.blur	Number , 默认值为 : 6	模糊指数
shadow.xoffset	Number , 默认值为 : 3	x偏移量
shadow.yoffset	Number , 默认值为 : 3	y偏移量

shapelink

样式名	赋值范围	说明
shapelink.type	String , 默认值为 : lineto 取值范围 : 'lineto', 'quadto', 'cubicto'	连线类型

shapenode - 多边形样式

样式名	赋值范围	说明
shapenode.closed	String , 默认值为 : false	是否闭合

vector - 图形样式

样式名	赋值范围	说明
vector.cap	String , 默认值为 : butt 取值范围 : butt, round, square	边框线端点样式
vector.deep	Number , 默认值为 : 0	深度
vector.fill	String , 默认值为 : true	是否填充
vector.fill.color	Color , 默认值为 : #CCCCFF	填充色
vector.gradient	String , 默认值为 : none 取值范围 : linear.east, linear.north, linear.northeast, linear.northwest, linear.south, linear.southeast, linear.southwest, linear.west, none, radial.center, radial.east,	填充渐变

	radial.north, radial.northeast, radial.northwest, radial.south, radial.southeast, radial.southwest, radial.west, spread.antidiagonal, spread.diagonal, spread.east, spread.horizontal, spread.north, spread.south, spread.vertical, spread.west	
vector.gradient.color	Color , 默认值为 : #FFFFFF	渐变色
vector.join	String , 默认值为 : miter 取值范围 : miter, round, bevel	边框线拐点样式
vector.outline.color	Color , 默认值为 : #5B5B5B	边框线颜色
vector.outline.width	Number , 默认值为 : -1	边框线宽度
vector.padding	Number , 默认值为 : 0	间隙
vector.padding.bottom	Number , 默认值为 : 0	底部间隙
vector.padding.left	Number , 默认值为 : 0	左边间隙
vector.padding.right	Number , 默认值为 : 0	右边间隙
vector.padding.top	Number , 默认值为 : 0	顶部间隙
vector.shape	String , 默认值为 : rectangle 取值范围 : rectangle, oval, roundrect, star, triangle, circle, hexagon, pentagon, diamond	图形形状

whole - 网元整体

样式名	赋值范围	说明
whole.alpha	Number , 默认值为 : 1	整体透明度