

TWaver Flex培训课程

——TWaver Flex基础

Serva Software LLC

课程内容

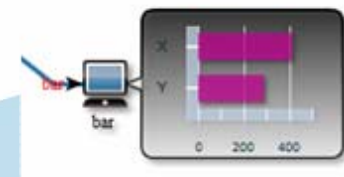
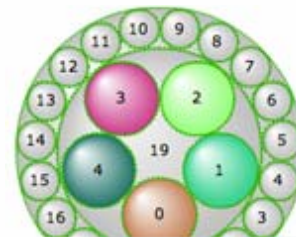
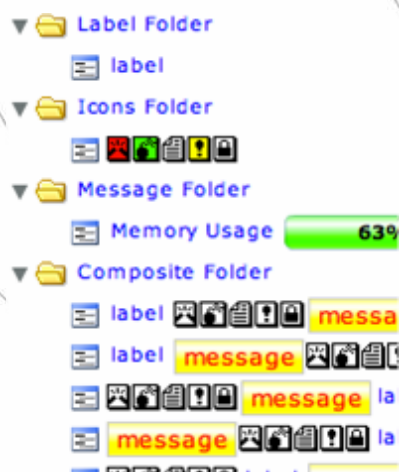
- TWaver是什么？
- Flex基础
 - Flex语言特性
 - Flex & Flash ...
- TWaver Flex基础
 - Hello TWaver!
 - MVC设计模式
 - 数据元素 / 数据容器 / 视图组件
- 与TWaver Java对照

TWaver是什么？

- 高效轻量的图形组件库
- 提供多种平台解决方案
- TWaver Flex产品包结构
- TWaver授权许可

TWaver是图形组件库

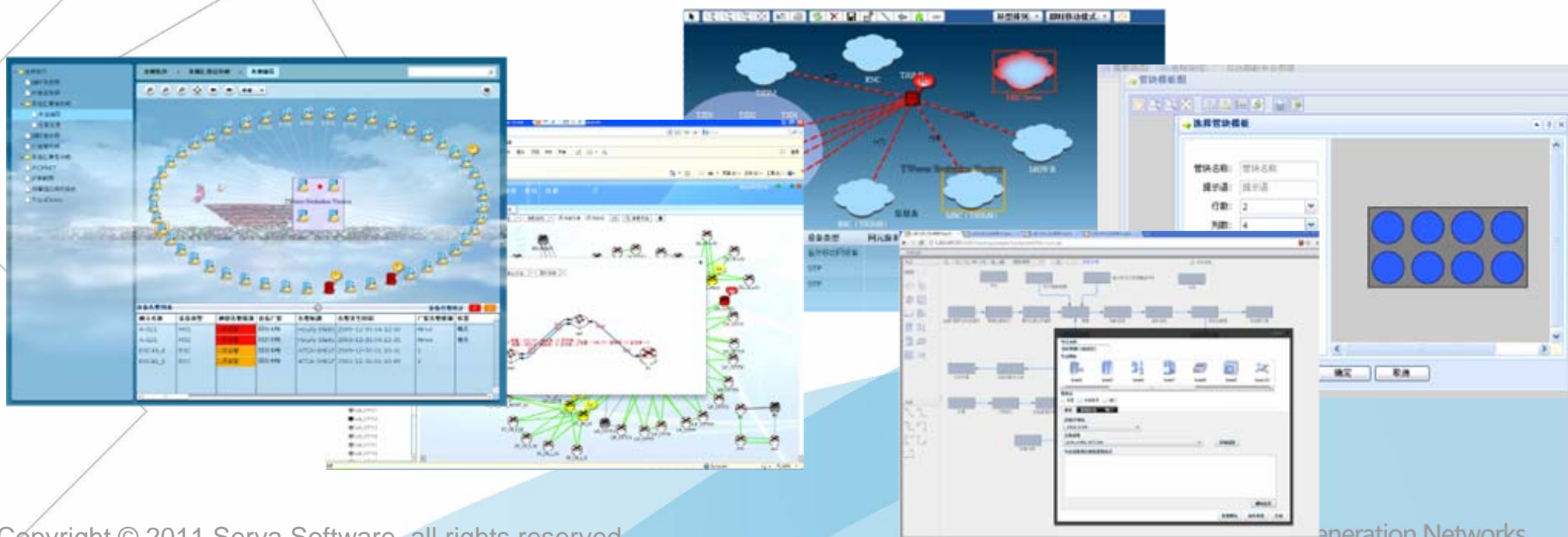
- TWaver关注数据的图形化展示
- TWaver面向开发人员，需要二次开发
- TWaver提供文档，许可，培训和技术支持



element	tree label	network label
twaver.Node	boy	boy
twaver.Node	iphone	iphone
twaver.Node	Internet	Internet
twaver.Node	computer	computer

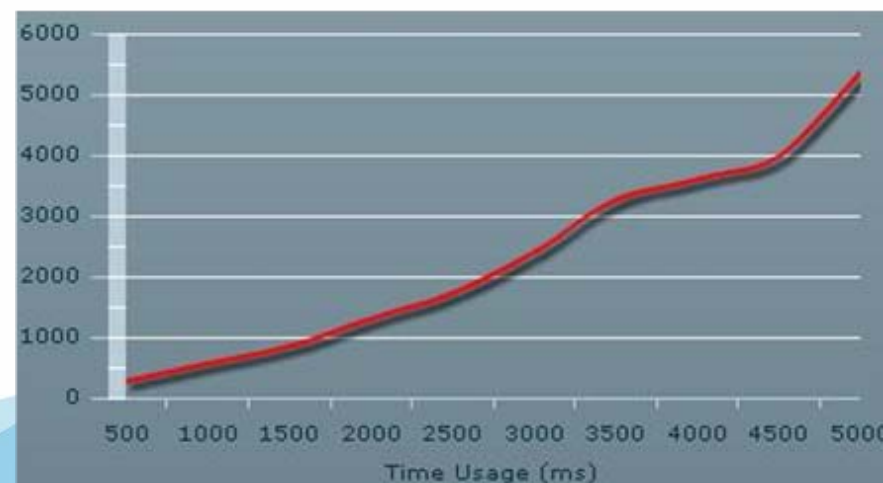
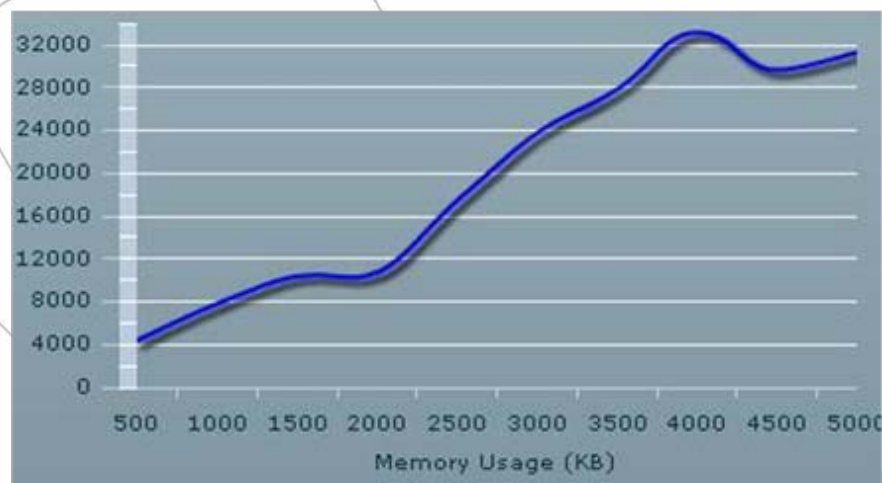
TWaver关注于电信网管

- 电信相关的业务模型（设备面板，告警传递.....）
- 积累了大量电信行业的应用案例
- 但不限于电信行业



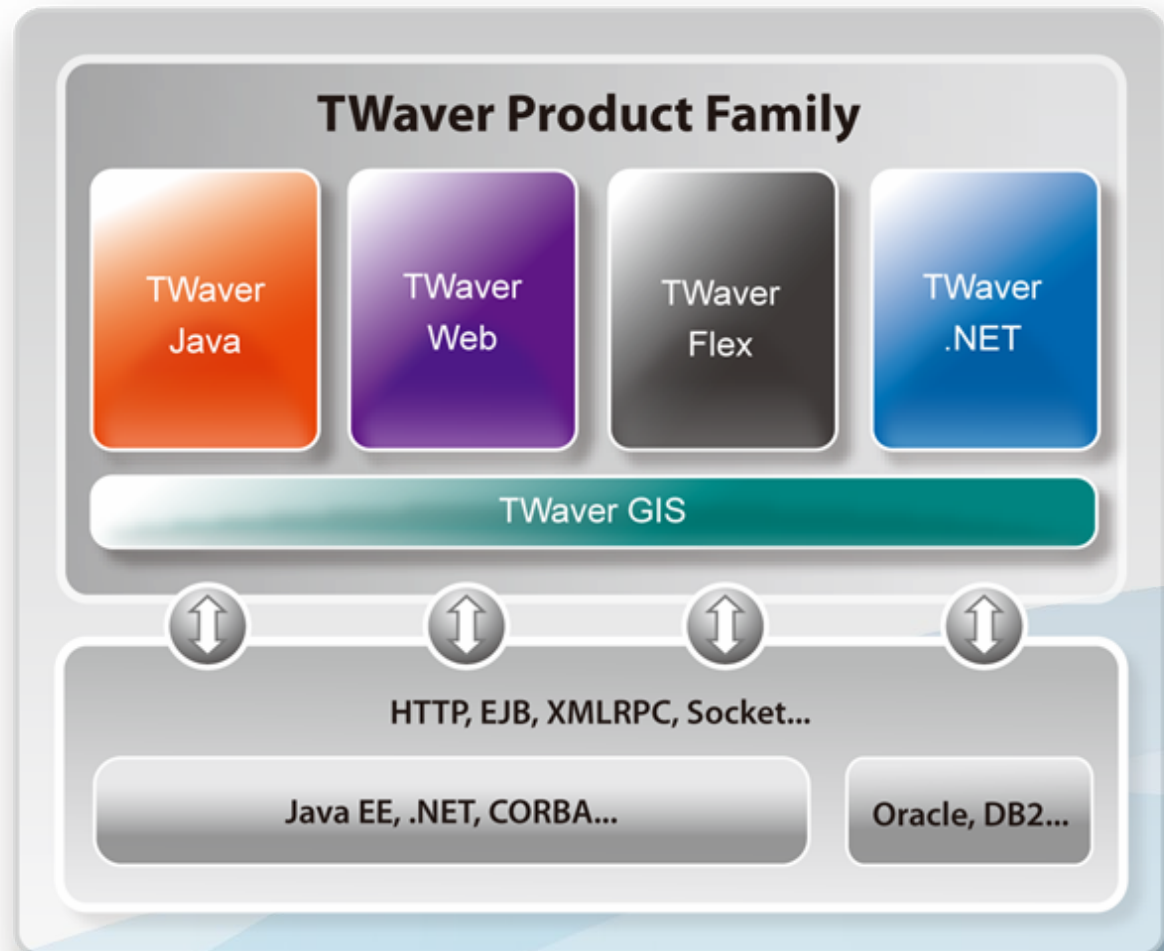
高效轻量的图形组件库

- TWaver.swc 475kb
- 拓扑图建议两千网元以内
- 两千个节点可在**1.5**秒内加载



提供多种平台解决方案

- Java
- Web
- **Flex**
- .NET
- HTML5...



丰富的客户应用案例



TWaver Flex产品包结构

TWaver™

twaver-flex-1.4	--
asdocs	--
Blue.swf	29 KB
demo.html	4 KB
demo.swf	2.1 MB
documents	--
TWaver Flex 1.4 Developer Guide.pdf	1.7 MB
lib	--
TWaver.swc	475 KB
License.txt	16 KB
README.txt	4 KB
src	--

API 文档

demo运行文件

开发手册

TWaver.swc

demo源代码

TWaver授权许可

- TWaver许可分三种，试用版，开发版，运行版。查看许可信息可在Flex Network界面使用 **Ctrl+Shift+T** 快捷键
- 试用版：可免费申请，用于前期预言或技术选型阶段，组件界面在五分钟后显示"TWaver Evaluation Version"水印
- 开发版：用于项目实际开发阶段，组件界面在两个小时会显示开发版水印
- 运行版：用于项目实际运行阶段，无水印

许可文件的使用

- **license.xml**是一个文本文件，包含许可授权信息
- 使用方式：
 - `<mx:XML source="demo/license.xml" id="licenseXML"/>`
 - `twaver.Utls.validateLicense(this.licenseXML);`

Flex基础

- Flex语言特性
 - get & set
 - .mxml & .as
- Flex & Flash
- 资源嵌入
- 其他

Flex语言特性

●ActionScript

- 源于脚本语言，使用与java更相近，是比较严谨的编译语言

- 函数重载

- 不支持函数重载，一个函数名对应一个函数，函数参数可设置默认值

- 脚本语言特性

- 保留了ECMAScript脚本语言的特性，支持arguments, function, prototype, call, apply, length, object['propertyA']，动态类(dynamic)

Flex中的get & set

```
/twaver/Data.as
package twaver{
    import flash.events.EventDispatcher;
    ...
    public class Data extends EventDispatcher implements IData{
        public function Data(name:String = null){
            this.name = name;
        }
        private var _name:String;
        public function set name(name:String):void{
            this._name = name;
        }
        public function get name():String{
            return this._name;
        }
    }
}
```

使用:

```
var data:Data = new Data();
data.name = "001";
trace(data.name);
```

.mxml & .as

- 主引导程序必须是.mxml文件
- .mxml文件最终都编译成.as类
- .mxml类可以用.as来实现
- .mxml和.as文件都可以定义类
- .mxml定义的类没法传入构造参数

Flex使用习惯

- var, function, “:”, 引号, get / set
- 没有Color, Double, Float, 用 int, Number
- children & toChildren
- forEach(callback)
- setStyle(name, value)
- addEventListener(type, listener, ...)
- CSS
- namespace, manifest.xml, xmlns

Flex & Flash

- Flex - open、 free
 - Flex SDK
 - “Flex is a highly productive, free, open source framework for building expressive mobile, web, and desktop applications.”
- Flash - closed、 paid for
 - Flash Builder
 - Flash Player

嵌入资源

- 嵌入的资源将被打包到swf文件
- 图片嵌入：
 - [Embed(source="resource/images/alarm.png")]
 - private static const alarmIcon:Class;
- 字体嵌入：
 - [Embed(source='Helvetica.ttf', fontName="demoFont",
 - advancedAntiAliasing="false", mimeType="application/x-font")]
 - private var demoFont:Class;
- XML嵌入：
 - <mx:XML source="demo/states.xml" id="statesXML"/>
- CSS文件嵌入：
 - <mx:Style source="Default.css" />

其他

- Graphics, Filter, BitmapData ...
- .swc / .swf / .air
- 本地域, 网络域, 安全域, crossdomain.xml
- ImageLoader, URLLoader, ModuleLoader
- HTTPService
- 调试, 发布

TWaver基础

- Hello TWaver
- MVC设计模式
- 数据元素 / 数据容器介绍
- 组件类型介绍

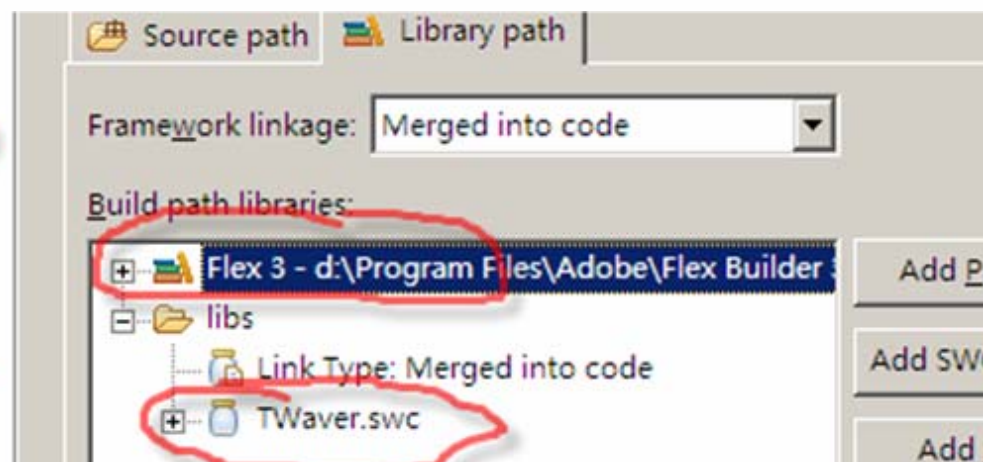
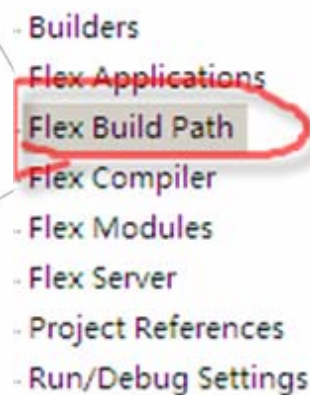
Hello TWaver

TWaver Flex 开发环境

- TWaver.swc
- Flex SDK 3.4.2+
- Flash Builder
- Flash Player 9+

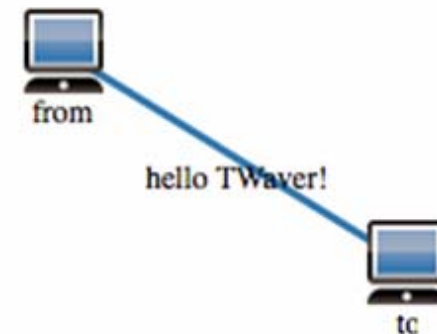
Hello TWaver

- 创建Flex Project
- 引入TWaver.swc
- 添加Flex SDK (3.4.2+)
- 参考开发手册“TWaver Flex开发环境”章节



Hello TWaver

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" pageTitle='Hello TWaver !'
  xmlns:twaver="http://www.servasoftware.com/2009/twaver/flex"
  applicationComplete="init()">
  <mx:Script>
    <![CDATA[
      import twaver.*;
      import twaver.network.Network;
      private function init():void{
        var box:ElementBox = network.elementBox;
        var from:Node = new Node();
        from.name = "from";
        from.location = new Point(20, 20);
        box.add(from);
        var to:Node = new Node();
        to.name = "to";
        to.location = new Point(150, 60);
        box.add(to);
        var link:Link = new Link(from,to);
        link.name = "hello TWaver!";
        box.add(link);
      }
    ]]>
  </mx:Script>
  <twaver:Network id="network" backgroundColor="0xffff" width="100%" height="100%"/>
</mx:Application>
```



增加Tree, Table

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:twaver="http://www.servasoftware.com/2009/twaver/flex"
  pageTitle="Hello TWaver!"
  applicationComplete="init()">
  <mx:Script>
    <![CDATA[
      import twaver.*;
      import twaver.network.Network;

      private var box:ElementBox;

      private function init():void{
        box=network.elementBox;
        var from:Node = new Node();
        from.name = "from";
        from.location = new Point(20, 20);
        box.add(from);
        var to:Node = new Node();
        to.name = "to";
        to.location = new Point(150, 100);
        box.add(to);
        var link:Link = new Link(from,to);
        link.name = "hello TWaver!";
        box.add(link);
      ]]>
    </mx:Script>
```

给树和表格设置数据容器

tree.dataBox=box;
table.dataBox=box;

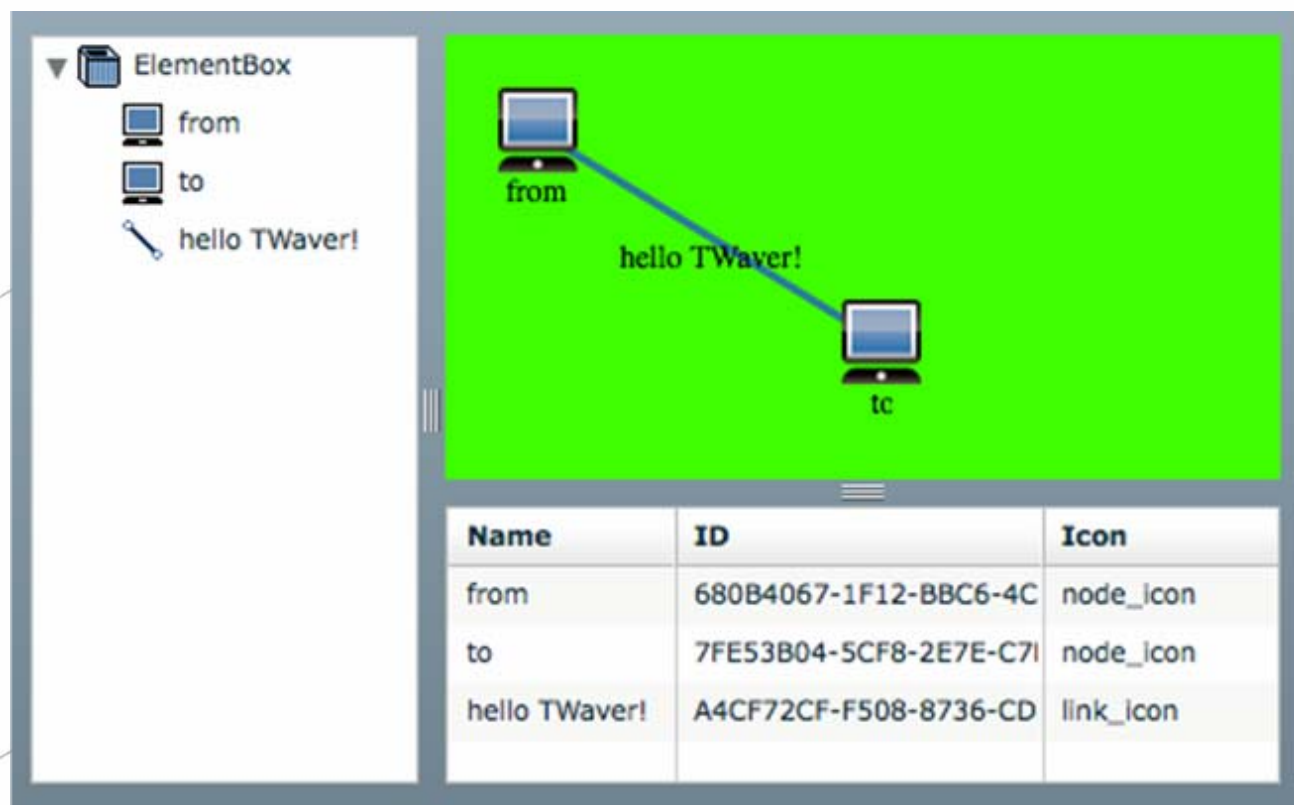
直接在MXML中配置表格

```
<mx:HDividedBox width="100%" height="100%">
  <twaver:Tree id="tree" width="30%" height="100%" />
  <mx:VDividedBox width="100%" height="100%">
    <twaver:Network id="network" backgroundColor="0x00ff00" width="100%" height="70%" />
    <twaver:Table width="100%" height="30%" id="table" editable="true">
      <twaver:columns>
        <twaver:TableColumn dataField="name" headerText="Name" />
        <twaver:TableColumn dataField="id" headerText="ID" />
        <twaver:TableColumn dataField="icon" headerText="Icon" />
      </twaver:columns>
```

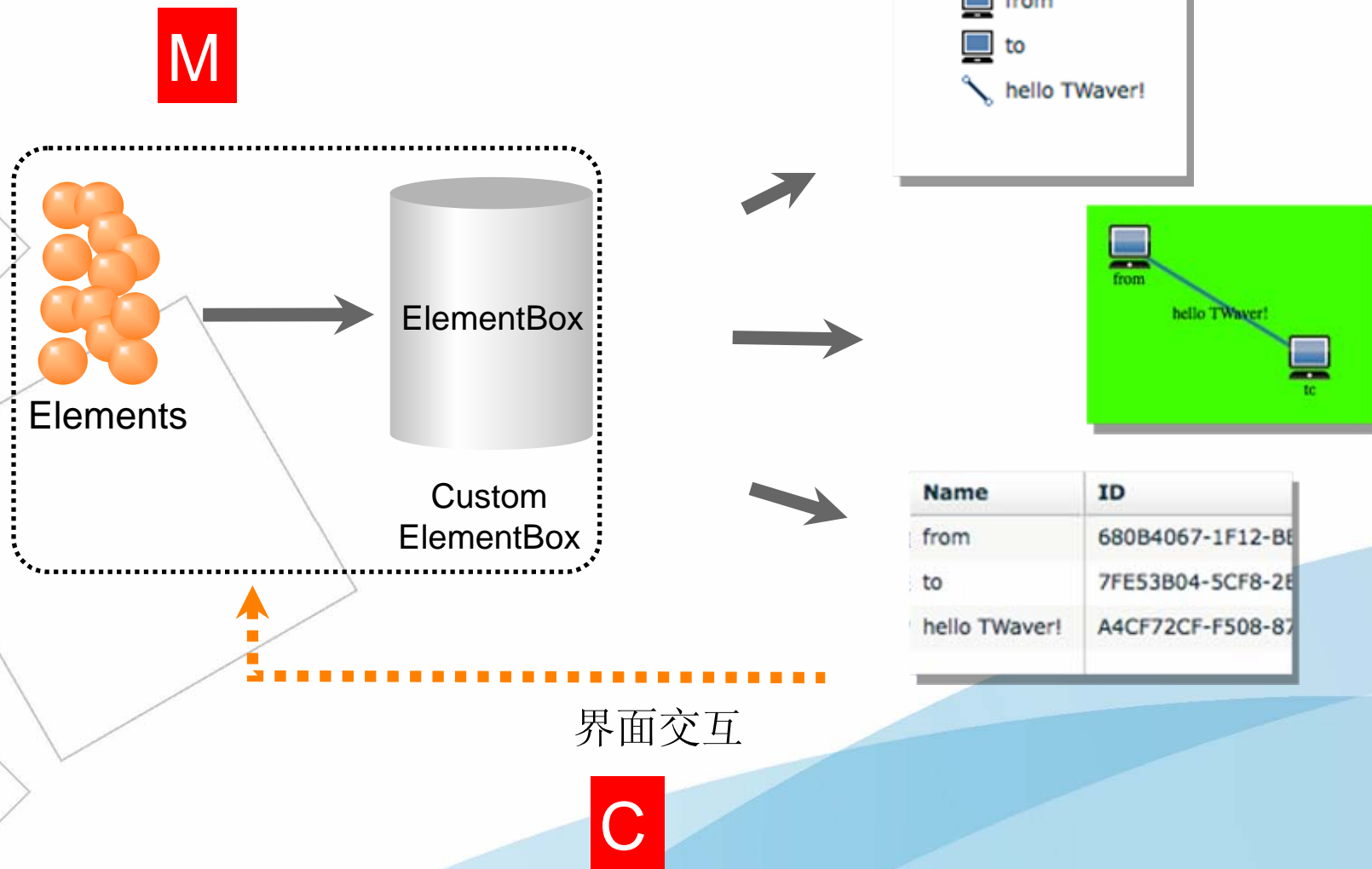
```
</twaver:Table>
</mx:VDividedBox>
</mx:HDividedBox>
</mx:Application>
```


Hello TWaver

运行界面

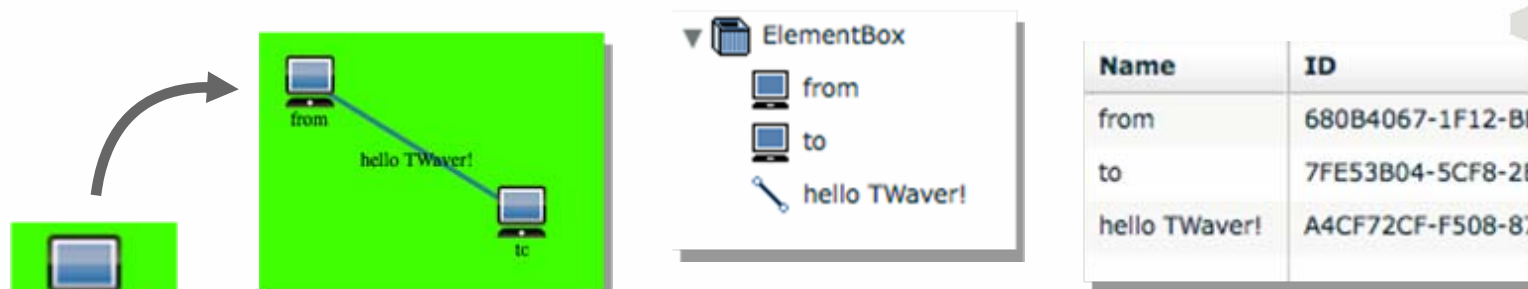


MVC设计模式



数据驱动

组件(V)



ElementUI

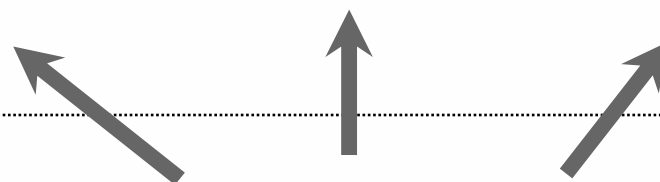
Element

模型(M)

Node
Link
Group
...



ElementBo
x



TWaverTM

事件机制

Element node.setStyle(Styles.INNER_COLOR, 0xFF0000)

dispatchPropertyChangeEvent

ElementBox

dataPropertyChangeListener

dispatchEvent

View

network, tree, chart ...

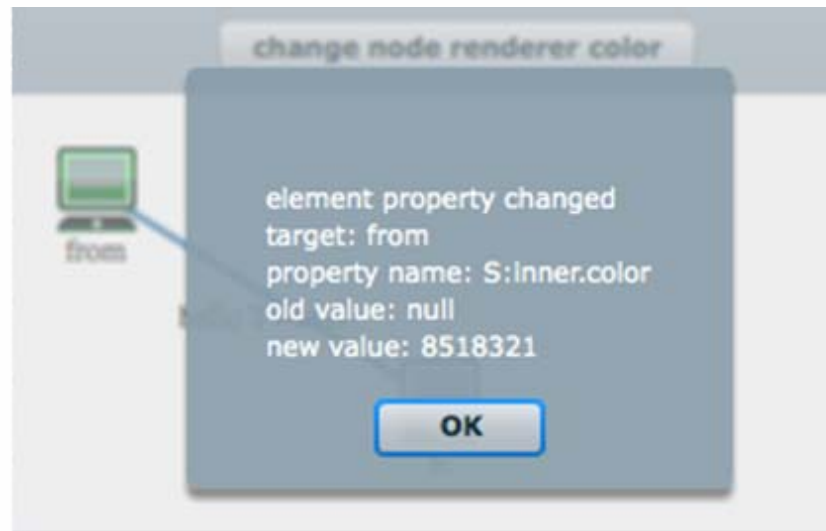
dataPropertyChangeListener

invalidate(element)



事件监听

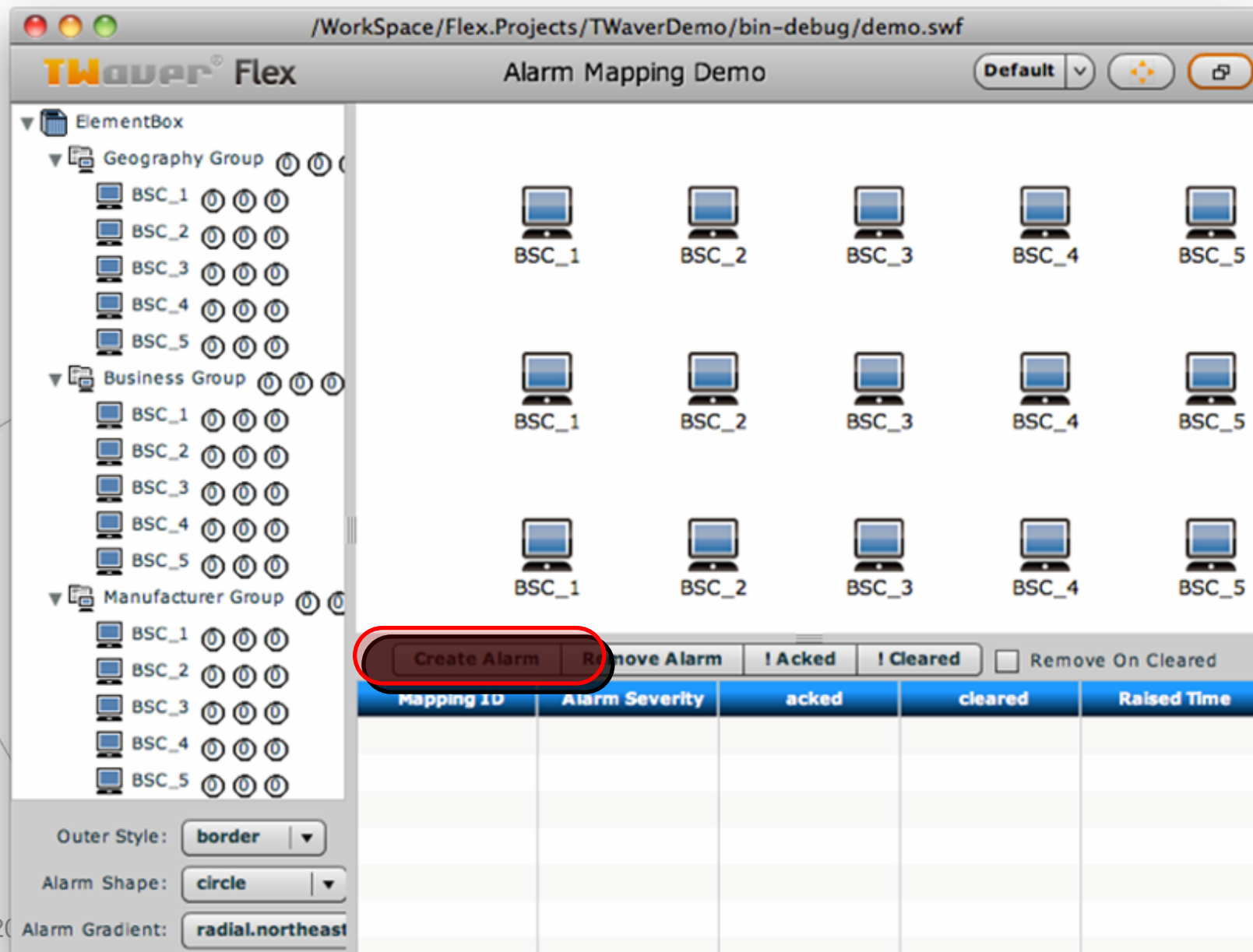
```
box.addDataPropertyChangeListener(function(evt:PropertyChangeEvent):void{  
    Alert.show("element property changed\ntarget: " + evt.source +  
        "\nproperty name: " + evt.property +  
        "\nold value: " + evt.oldValue +  
        "\nnew value: " + evt.newValue);  
});
```



事件监听

Target	Listener
DataBox	addDataBoxChangeListener addDataPropertyChangeListener addHierarchyChangeListener addPropertyChangeListener
ElementBox	extends DataBox addIndexChangeListener
AlarmBox	extends DataBox
LayerBox	extends DataBox
SelectionModel	addSelectionChangeListener
Network	addInteractionListener addPropertyChangeListener
AlarmSeverity	addAlarmSeverityChangeListener

组件联动



组件联动

Alarm Mapping Demo

Default

Geography Group

BSC_1 BSC_2 BSC_3 BSC_4 BSC_5

Business Group

BSC_1 BSC_2 BSC_3 BSC_4 BSC_5

Manufacturer Group

BSC_1 BSC_2 BSC_3 BSC_4 BSC_5

Create Alarm Remove Alarm ! Acked ! Cleared ☐ Remove On Cleared

Mapping ID	Alarm Severity	acked	cleared	Raised Time
4	Major	<input type="checkbox"/>	<input type="checkbox"/>	Thu Jun 9 16:51

Outer Style: border

Alarm Shape: circle

Alarm Gradient: radial.northeast

数据元素 / 数据容器

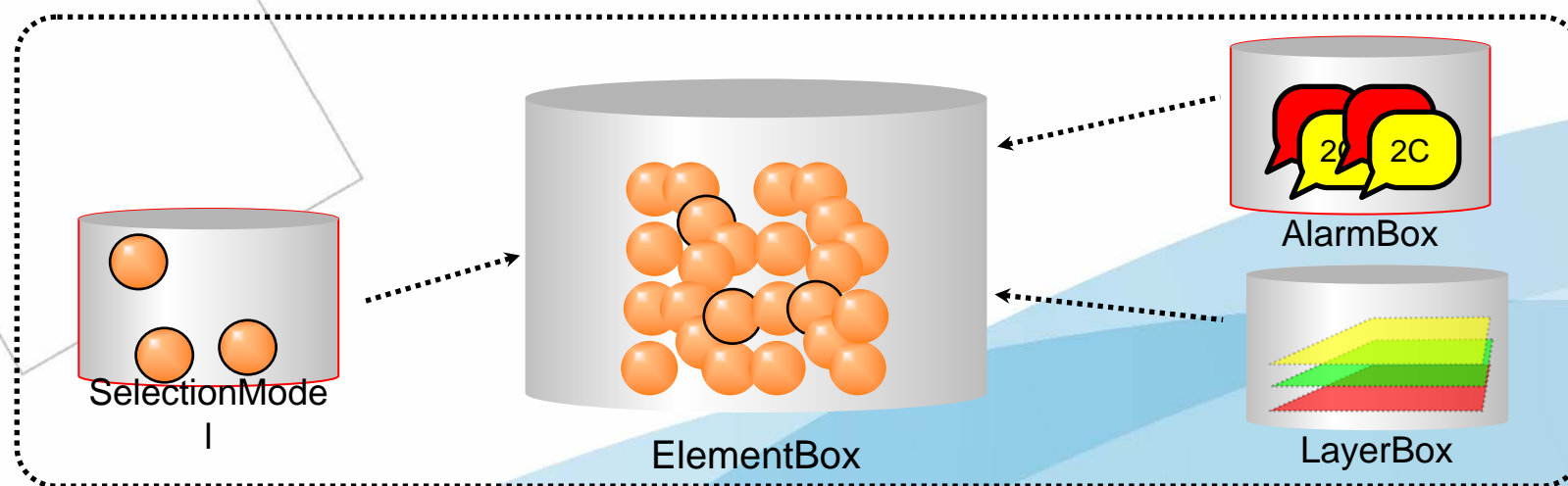
数据元素：数据的基本单位

数据容器：管理数据元素集合的容器，提供对数据元素的增减操作，监听数据元素的属性变化

数据元素	数据容器
IData	DataBox
IElement	ElementBox
IAlarm	AlarmBox
ILayer	LayerBox

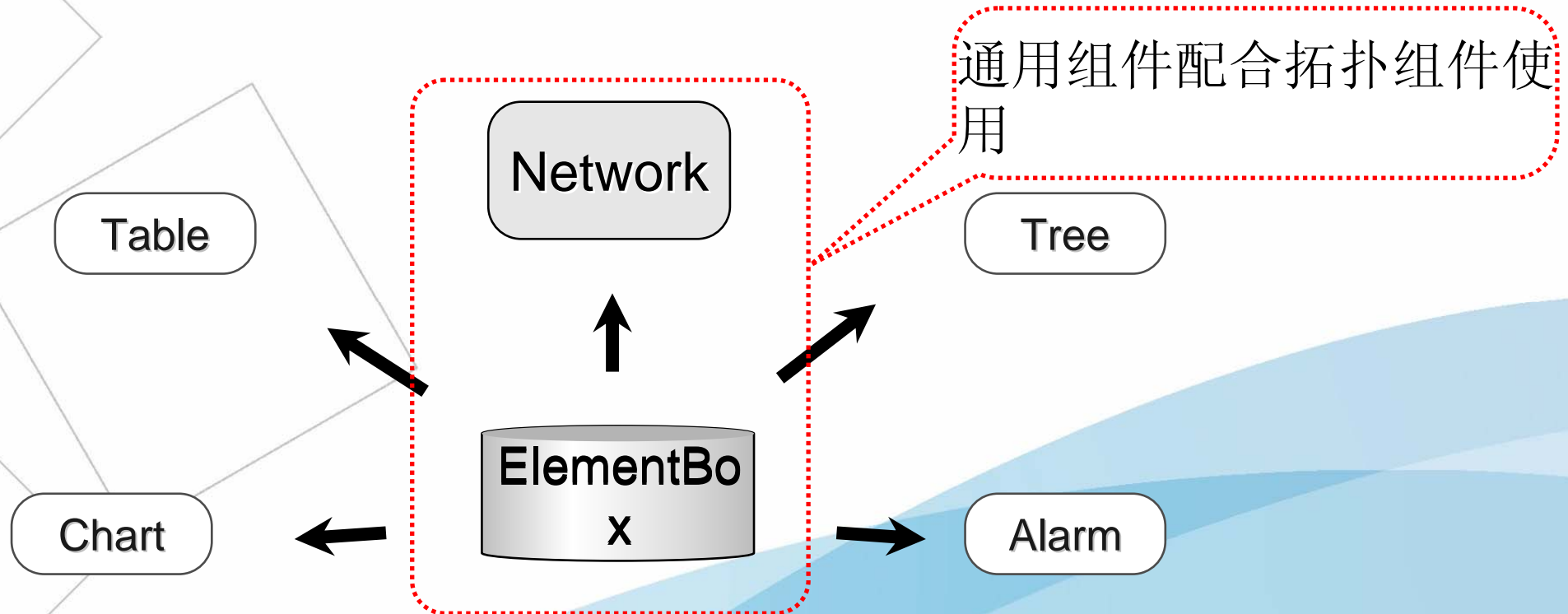
数据容器之间关系

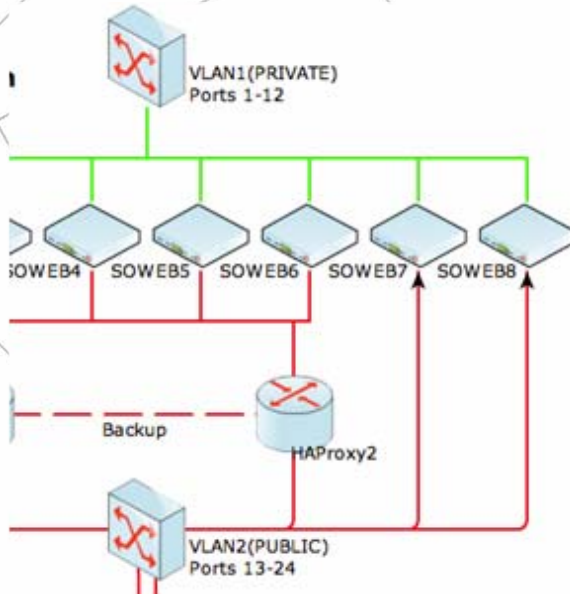
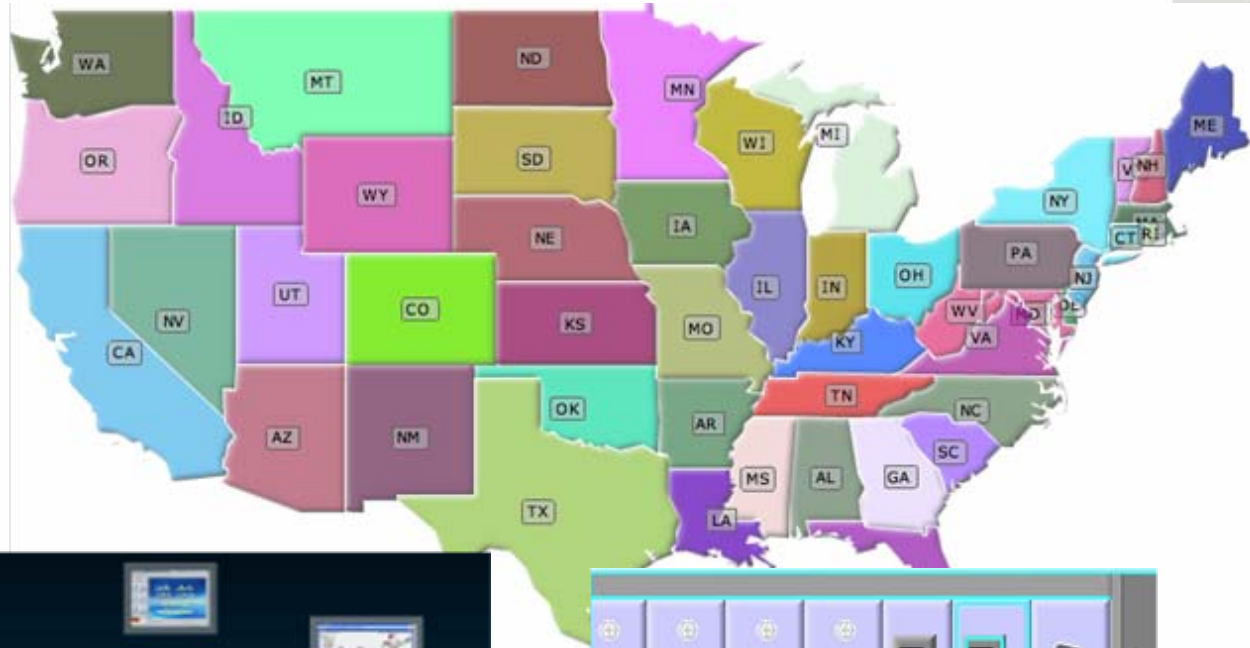
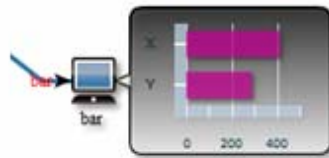
- **ElementBox**是管理网元的数据容器，同时他也包含了告警容器，图层容器，选中模型等。



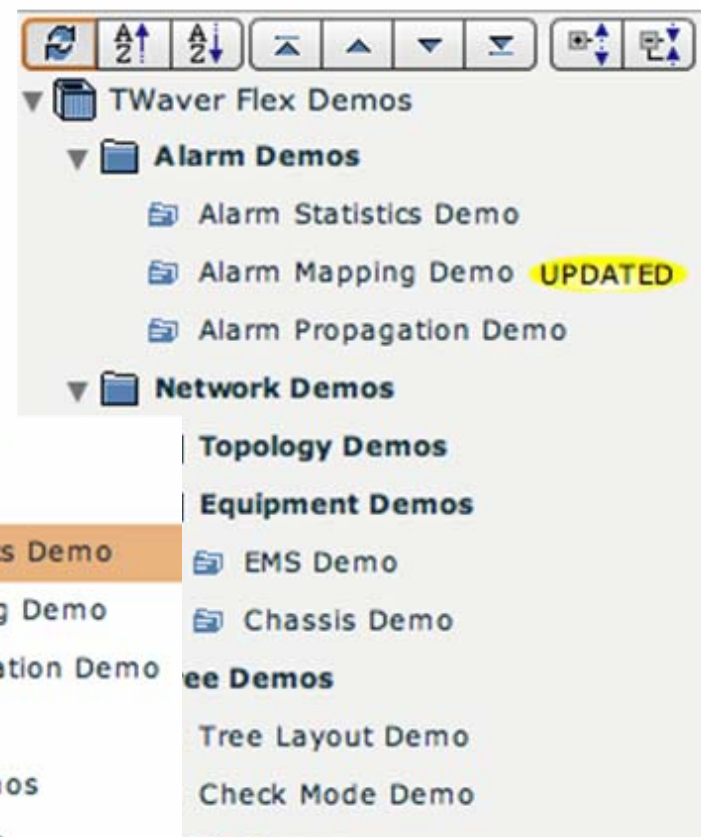
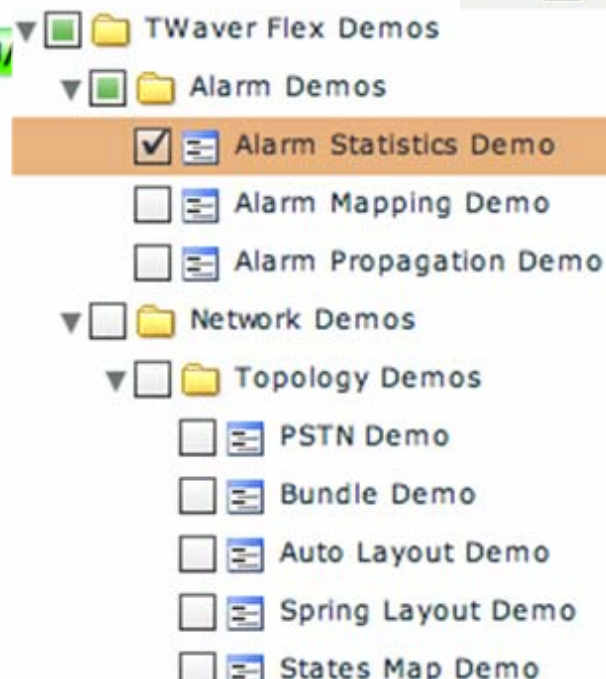
组件类型

- 拓扑组件: Network
- 通用组件: Tree, Table, Chart ...















Tree



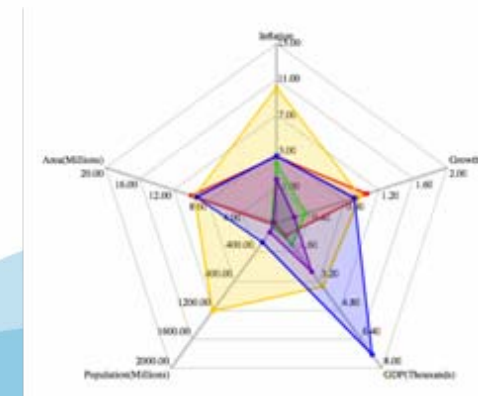
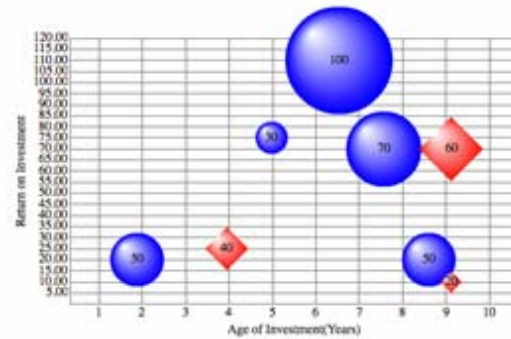
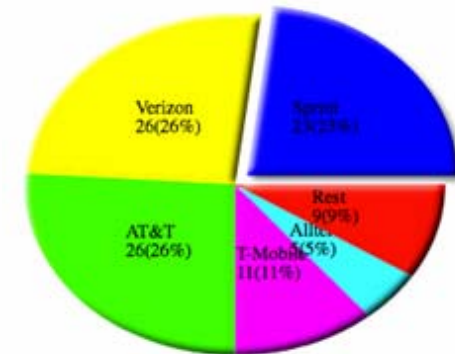
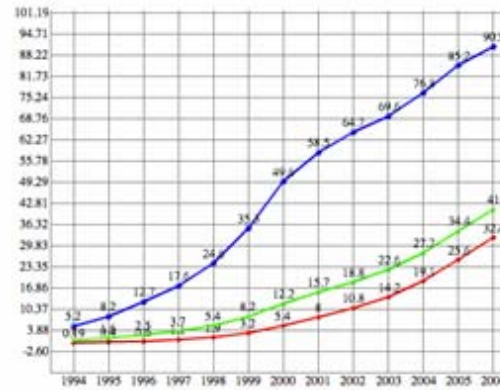
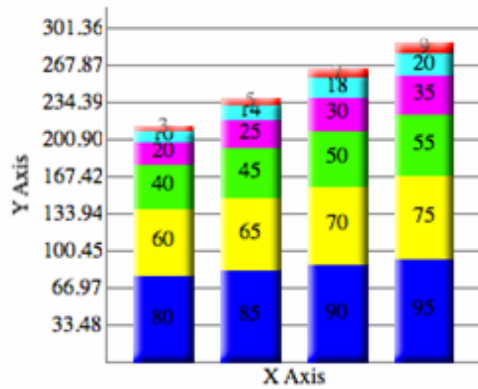
Table

state	employed	unemploy	male	female
District of Columbia	303994	23442	282970	323930
Mississippi	1028773	94712	1230617	1342599
Alabama	1741794	128587	1936162	2104425
New York	8498119	636280	8739138	9496769
Pennsylvania	5434532	344795	5694265	6187378

Mapping ID	Alarm Severity	acked	cleared	Raised Time
1	Indeterminate	<input type="checkbox"/>	<input type="checkbox"/>	Fri Jun 10 14:25:51 GMT+0800 2011
2	Warning	<input type="checkbox"/>	<input type="checkbox"/>	Fri Jun 10 14:25:51 GMT+0800 2011
3	Minor	<input type="checkbox"/>	<input type="checkbox"/>	Fri Jun 10 14:25:51 GMT+0800 2011
4		<input type="checkbox"/>	<input type="checkbox"/>	Fri Jun 10 14:25:51 GMT+0800 2011
5	Cleared	<input type="checkbox"/>	<input type="checkbox"/>	Fri Jun 10 14:25:51 GMT+0800 2011
5	Indeterminate	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Fri Jun 10 14:25:51 GMT+0800 2011
4	Warning	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Fri Jun 10 14:25:51 GMT+0800 2011
3	Minor	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Fri Jun 10 14:25:51 GMT+0800 2011
2	Minor	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Fri Jun 10 14:25:51 GMT+0800 2011
1	Critical	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Fri Jun 10 14:25:51 GMT+0800 2011

element	tree label	network label
 twaver.Node	boy	boy
 twaver.Node	iphone	iphone
 twaver.Node	internet	internet
 twaver.Node	computer	computer
 twaver.Node	girl	girl
 twaver.Link	boy --> iphone	
 twaver.Link	boy -- iphone	
 twaver.Link	boy <-- iphone	
 twaver.Link	iphone --> internet	
 twaver.Link	iphone -- internet	

Chart



与TWaver Java对照

- TWaver Flex与TWaver Java使用上有相似之处，有很多相似的类和方法
- 主要类对照
- 网元属性设置对照
- 拓扑图组件使用对照

主要类对照

TWaver Flex	TWaver Java	说明
ElementBox	TDataBox	网元容器
AlarmBox	AlarmModel	告警容器
LayerBox	LayerModel	图层容器
Network	TNetwork	拓扑图
Tree	TTree	树图
Table	TElementTable	表格

网元属性设置对照

TWaver Flex	TWaver Java	说明
<code>node.name = "001"</code>	<code>node.setName("001")</code>	属性
<code>node.setStyle</code>	<code>node.putClientProperty</code>	样式属性
<code>node.setClient</code>	<code>node.putUserProperty</code>	用户属性

拓扑图使用对照

TWaver Flex	TWaver Java	说明
alarmLabelFunction	alarmLabelGenerator	告警冒泡标签
visibleFunction	visibleFilter	可见过滤器
movableFunction	movableFilter	可移动过滤器
editableFunction	elementLabelEditableFilter	可编辑过滤器
labelFunction	elementLabelGenerator	网元标签文本生成器
toolTipFunction	elementToolTipTextGenerator	提示文本生成器
innerColorFunction	elementBodyColorGenerator	网元体颜色渲染
outerColorFunction	elementOutlineColorGenerator	网元边框颜色渲染
selectColorFunction	elementSelectColorGenerator	选中颜色生成器
alarmFillColorFunction	alarmColorGenerator	告警冒泡颜色



- 论坛: twaver.servasoft.com/forum
- MSN: twavercn@hotmail.com
- 邮件列表: twaver-news@servasoft.com

TWaver Flex培训课程

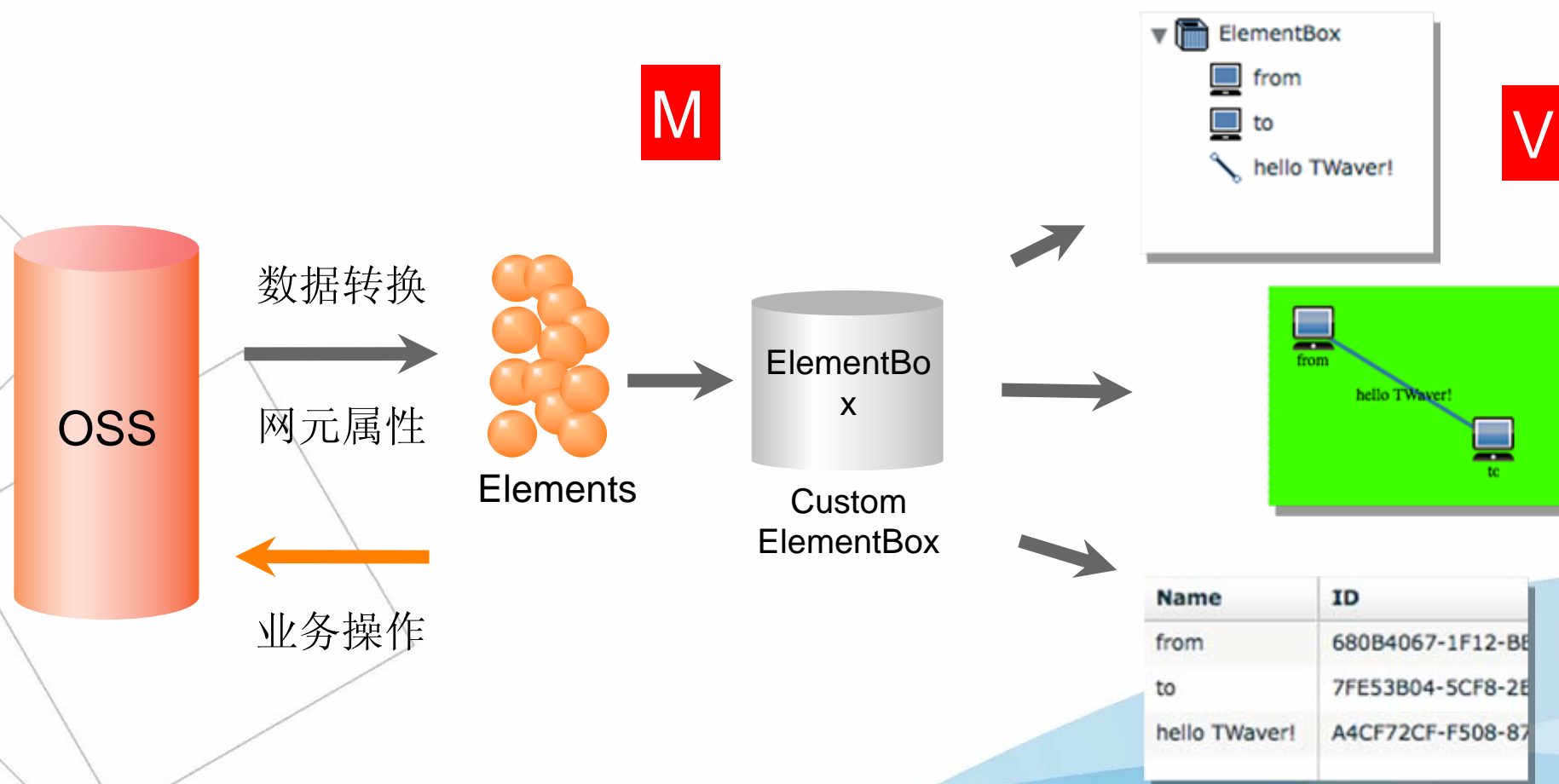
——TWaver Flex核心组件的使用

Serva Software LLC

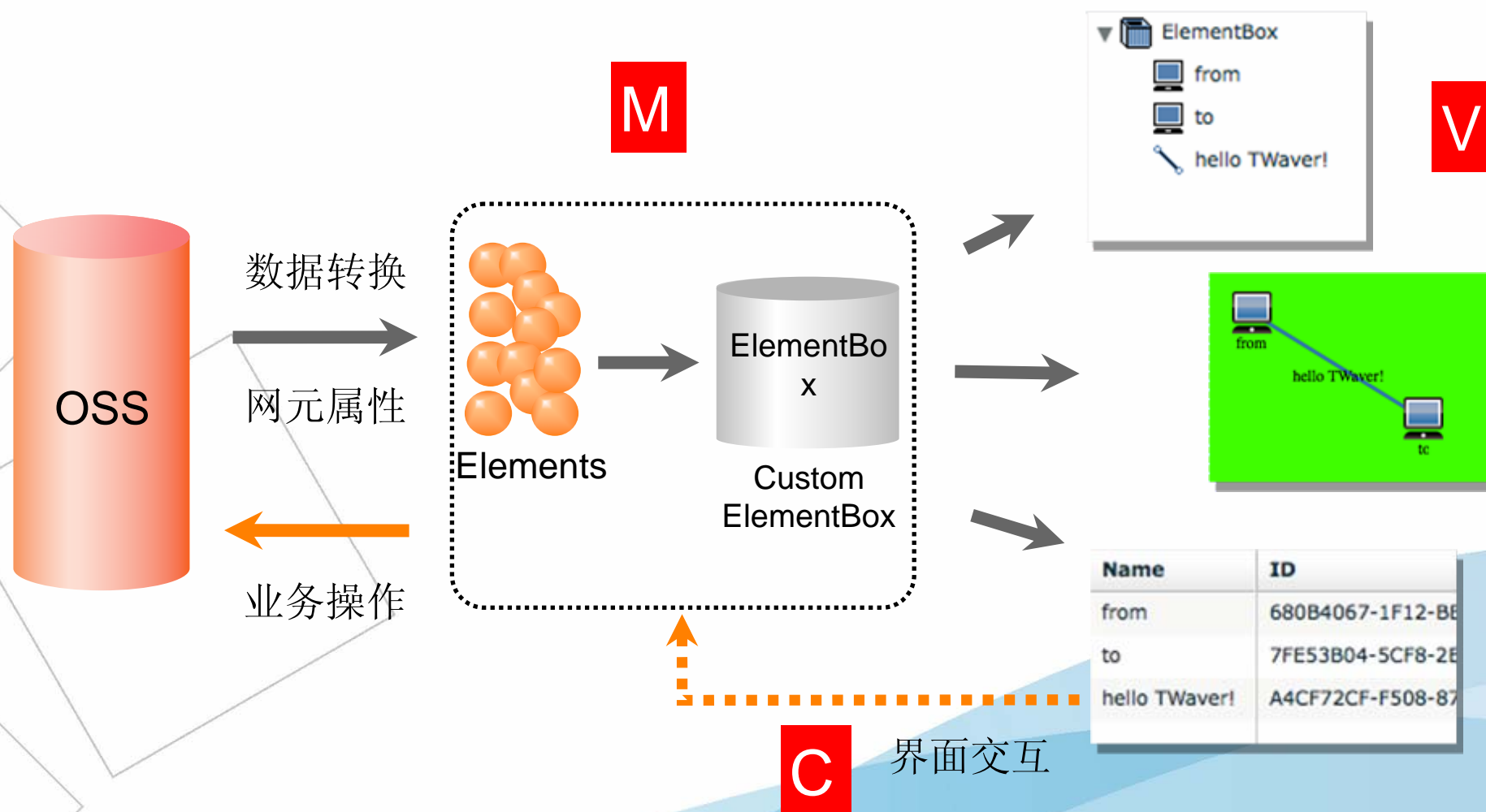
课程内容

- 开发流程
- DataBox & ElementBox的使用
- 预定义网元
- Network的使用

TWaver Flex开发流程



TWaver Flex开发流程



TWaver™

TWaver Flex开发流程示例

```
box = network.elementBox;
```

```
//数据采集
```

```
var devices:Array = getDevicesFromOSS();
```

```
var relationships:Array = getDevicesRelationshipFromOSS();
```

```
//数据转换
```

```
translateToTWaverNode(box, devices, relationships);
```

```
var layouter:AutoLayouter = new AutoLayouter(network);
```

```
layouter.animate = false;
```

```
layouter.doLayout(Consts.LAYOUT_SYMMETRY);
```

```
//添加交互
```

```
network.addListener(function(evt:InteractionEvent):void{
```

```
    if(evt.kind != InteractionEvent.DOUBLE_CLICK_ELEMENT){
```

```
        return;
```

```
    }
```

```
    var element:IElement = evt.element;
```

```
    InputDialog.show(network, "Rename", element.name, function(text:String):void{
```

```
        element.name = text;
```

```
        //入库操作
```

```
    }, evt.mouseEvent.stageX, evt.mouseEvent.stageY);
```

```
});
```

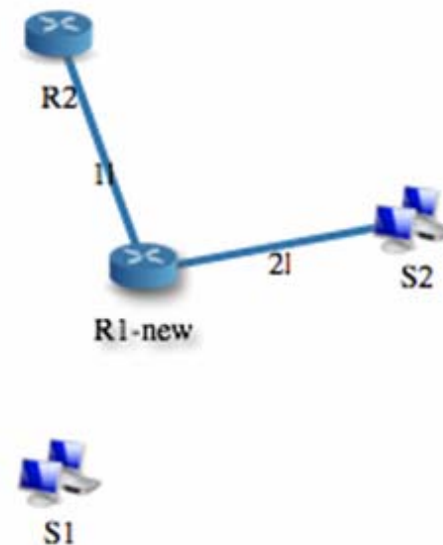
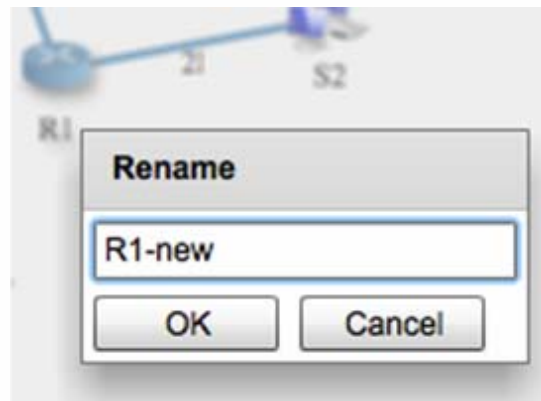
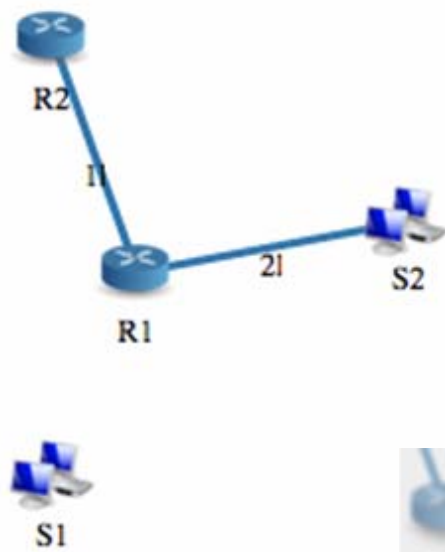
TWaver Flex开发流程示例

```
private function getDevicesFromOSS():Array{
    return [{name:'R1', type: 'router'}, {name:'R2', type: 'router'}, {name:'S1', type: 'device'}, {name:'S2', type: 'device'}] ;
}

private function getDevicesRelationshipFromOSS():Array{
    return [{name:'1I', from:'R1', to:'R2'}, {name:'2I', from:'R1', to:'S2'}];
}

private function translateToTWaverNode(box:ElementBox, devices:Array, relationships:Array):void{
    devices.forEach(function(device:*, index:int, arr:*)void{
        var node:Node = new Node(device.name);
        node.name = device.name;
        node.image = device.type;
        box.add(node);
    });
    relationships.forEach(function(relationship:*, index:int, arr:*)void{
        var link:Link = new Link(box.getElementByID(relationship.from) as Node, box.getElementByID(relationship.to)
as Node);
        link.name = relationship.name;
        box.add(link);
    });
}
```

TWaver Flex开发流程示例

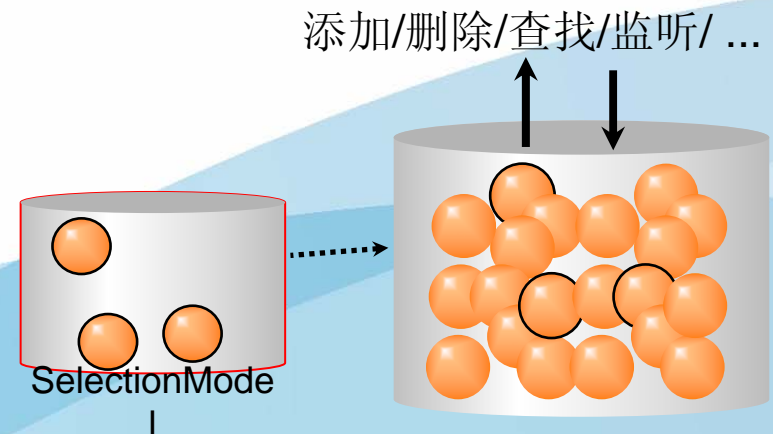
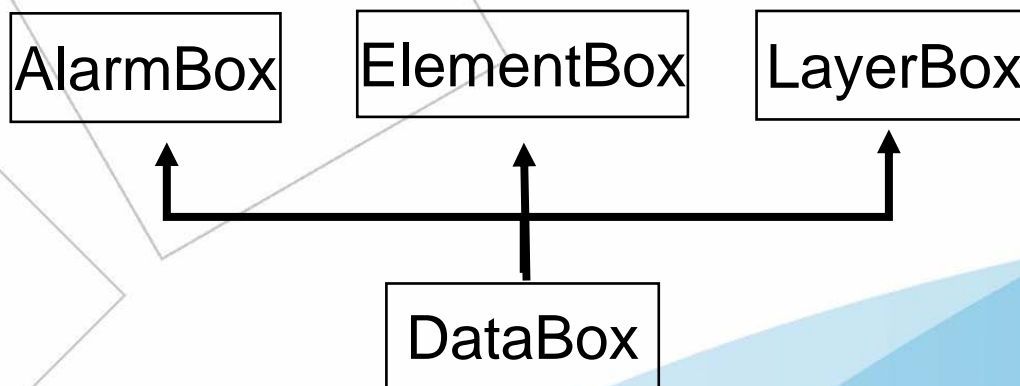


DataBox的使用

DataBox: 用于管理数据元素的容器。

- 提供添加, 删除, 修改, 遍历数据元素的操作;
- 监听数据属性变化, 容器变化, 层次变化等事件;
- 是ElementBox、AlarmBox、LayerBox的基类
- 包含选中模型

DataBox#**public function get** selectionModel():SelectionModel



数据的增减清除

- 增加数据

public function add(data:IData, index:int = -1):**void**

- 删除数据

public function remove(data:IData):**void**

public function removeSelection():**void**

public function removeByID(id:Object):**void**

- 清空容器

public function clear():**void**

数据的访问

- 获取数据

public function getDataByID(id:Object):IData

public function get datas():ICollection

public function toDatas(matchFunction:Function = **null**):ICollection

public function get roots():ICollection

public function getSiblings(data:IData):ICollection

- 遍历数据

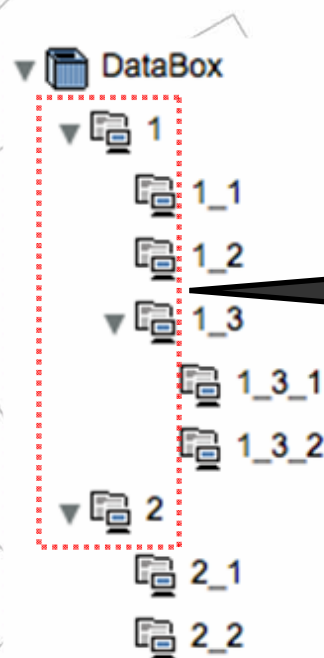
public function forEach(callbackFunction:Function):**void**

public function forEachByDepthFirst(callbackFunction:Function, data:IData = **null**):**void**

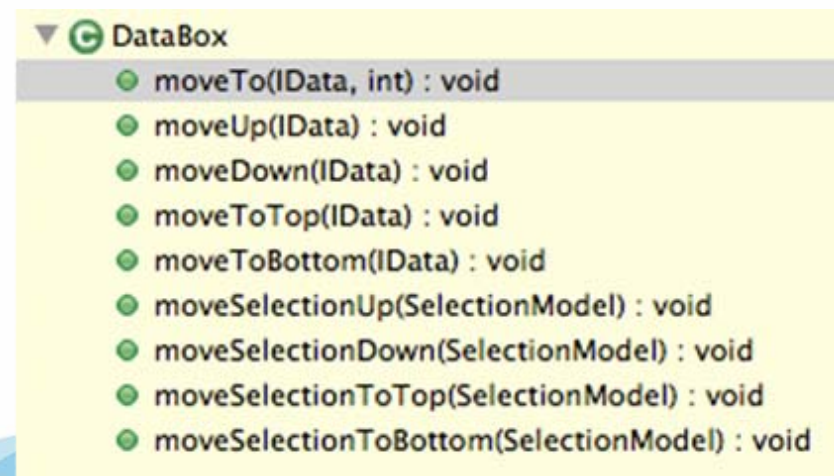
public function forEachByBreadthFirst(callbackFunction:Function, data:IData = **null**):**void**

数据的顺序关系

- DataBox中的数据通过父子关系形成层次结构
- move***方法可以调整数据在当层的前后顺序



节点1, 2位于最根层
(roots)



快速查找器

QuickFinder:

用于查找具有某个属性的数据元素，如找出所有名称为“PC”的网元：
`var result:Array = new QuickFinder(box, "name").find("PC");`

创建快速查找器:

```
public function QuickFinder(dataBox:DataBox, propertyName:String,  
    propertyType:String = "accessor",  
    valueFunction:Function = null,  
    filterFunction:Function = null)
```

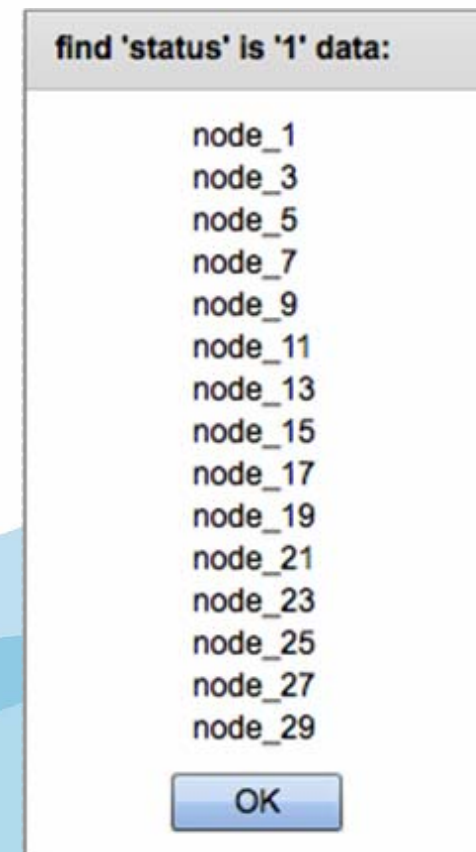
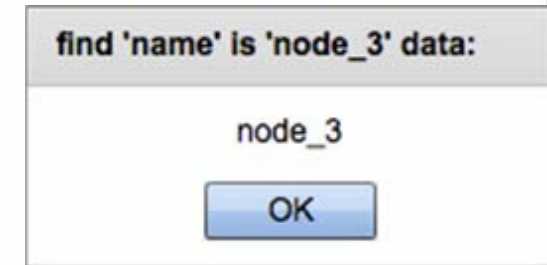
查找器种类（propertyType）

- `Consts.PROPERTY_TYPE_ACCESSOR` — 通过属性查找
- `Consts.PROPERTY_TYPE_CLIENT` — 通过client查找
- `Consts.PROPERTY_TYPE_STYLE` — 通过style查找

网元的快速查找示例

```
private function init():void{
    var box:DataBox = new DataBox();
    for (var i:int = 0; i < 30; i++) {
        var data:Data = new Data();
        data.name = "node_" + i;
        data.setClient("status", i%2);
        box.add(data);
    }
    var nameFinder:QuickFinder = new QuickFinder(box, 'name');
    var result:Array = nameFinder.find("node_3");
    alertArray(result, function():void{
        var clientFinder:QuickFinder = new QuickFinder(box, "status",
        Consts.PROPERTY_TYPE_CLIENT);
        alertArray(clientFinder.find(1), null);
    });
}

private function alertArray(result:Array, closeHandler:Function):void{
    var sResult:String = "";
    result.forEach(function(item:IData, index:*, arr:*)void{
        sResult += item.name + '\n';
    });
    Alert.show(sResult, null, 4, null, closeHandler);
}
```



DataBox中的监听器

添加监听器: DataBox#add***Listener

删除监听: DataBox#remove***Listener

```
box.addDataPropertyChangeListener(function(evt:PropertyChangeEvent):void{  
    trace((evt.source as Data).name + "'s property changed");  
});
```

监听器	作用
addDataBoxChangeListener	监听数据的添加删除, 清除
addPropertyChangeListener	监听dataBox属性变化
addDataPropertyChangeListener	监听数据属性变化
addHierarchyChangeListener	监听数据层次变化

监听器使用示例

```
private function init():void{
    var box:DataBox = new DataBox();
    box.addDataBoxChangeListener(function(evt:DataBoxChangeEvent):void{
        trace("data " + evt.kind + ": " + evt.data.name);
    });

    box.addDataPropertyChangeListener(function(evt:PropertyChangeEvent):void{
        trace((evt.source as Data).name + "'s property changed");
    });

    var data:Data = new Data();
    data.name = '001';
    //添加元素到容器
    box.add(data);
    //修改元素属性
    data.name = '002';
    //从容器中删除元素
    box.remove(data);
}
```

输出结果:

```
data add: 001
002's property changed
data remove: 002
```

DataBox的导入导出

- DataBox中元素数据可以导出导入xml
- 导出的xml包含所有元素属性和dataBox自身属性
- 导入导出通过XMLSerializer类实现

DataBox的导入导出

●XMLSerializer（导出 / 导入）

- **public function** XMLSerializer(dataBox:DataBox, settings:SerializationSettings = **null**)
- **public function** serialize():String
- **public function** deserialize(xmlString:String, rootParent:IData = **null**):**void**

●SerializationSettings（导入导出设置）

- **public function** registerProperty(property:String, type:String, cdata:Boolean=**false**):**void**
- **public function** registerStyle(styleProp:String, type:String, cdata:Boolean=**false**):**void**
- **public function** registerClient(clientProp:String, type:String, cdata:Boolean=**false**):**void**

DataBox导出示例

```
var box:DataBox = new DataBox();  
box.name = 'box001';
```

```
var data:Data = new Data();  
data.name = '001';  
data.setClient('age', 27);  
box.add(data);  
var child:Data = new Data();  
child.name = '001-1';  
child.parent = data;  
box.add(child);
```

```
var settings:SerializationSettings = new SerializationSettings();  
settings.registerClient('age', Consts.TYPE_INT);  
settings.registerProperty('name', Consts.TYPE_STRING);  
settings.registerProperty("parent", Consts.TYPE_DATA);
```

```
var serializer:XMLSerializer = new XMLSerializer(box, settings);  
trace(serializer.serialize());
```

输出:

```
<twaver v='1.4' p='flex'>  
<dataBox type='twaver.DataBox'>  
  <p n='name'>box001</p>  
</dataBox>  
<data type='twaver.Data' ref='0'>  
  <c n='age'>27</c>  
  <p n='name'>001</p>  
</data>  
<data type='twaver.Data' ref='1'>  
  <p n='name'>001-1</p>  
  <p n='parent' ref='0'/>  
</data>  
</twaver>
```

Twaver™

DataBox导入示例

```
private var xml:XML=<twaver v='1.4' p='flex'>
<dataBox type='twaver.DataBox'>
  <p n='name'>box001</p>
</dataBox>
<data type='twaver.Data' ref='0'>
  <c n='age'>27</c>
  <p n='name'>001</p>
</data>
<data type='twaver.Data' ref='1'>
  <p n='name'>001-1</p>
  <p n='parent' ref='0'>
</data>
</twaver>
```

```
private function init():void{
  var box:DataBox = new DataBox();
```

```
  var settings:SerializationSettings = new SerializationSettings();
  settings.registerClient('age', Consts.TYPE_INT);
  settings.registerProperty('name', Consts.TYPE_STRING);
  settings.registerProperty("parent", Consts.TYPE_DATA);
```

```
  var serializer:XMLSerializer = new XMLSerializer(box, settings);
```

```
  serializer.deserialize(xml.toXMLString());
```

```
  trace('box name is ' + box.name);
  trace('box count is ' + box.count);
  box.forEach(function(data:Data):void{
    trace('data name is ' + data.name);
    if(data.getClient("age")){
      trace('age is ' + data.getClient('age'));
    }
    if(data.parent){
      trace('parent is ' + data.parent.name);
    }
  })
}
```

输出:

```
box name is box001
box count is 2
data name is 001
age is 27
data name is 001-1
parent is 001
```

Twaver™

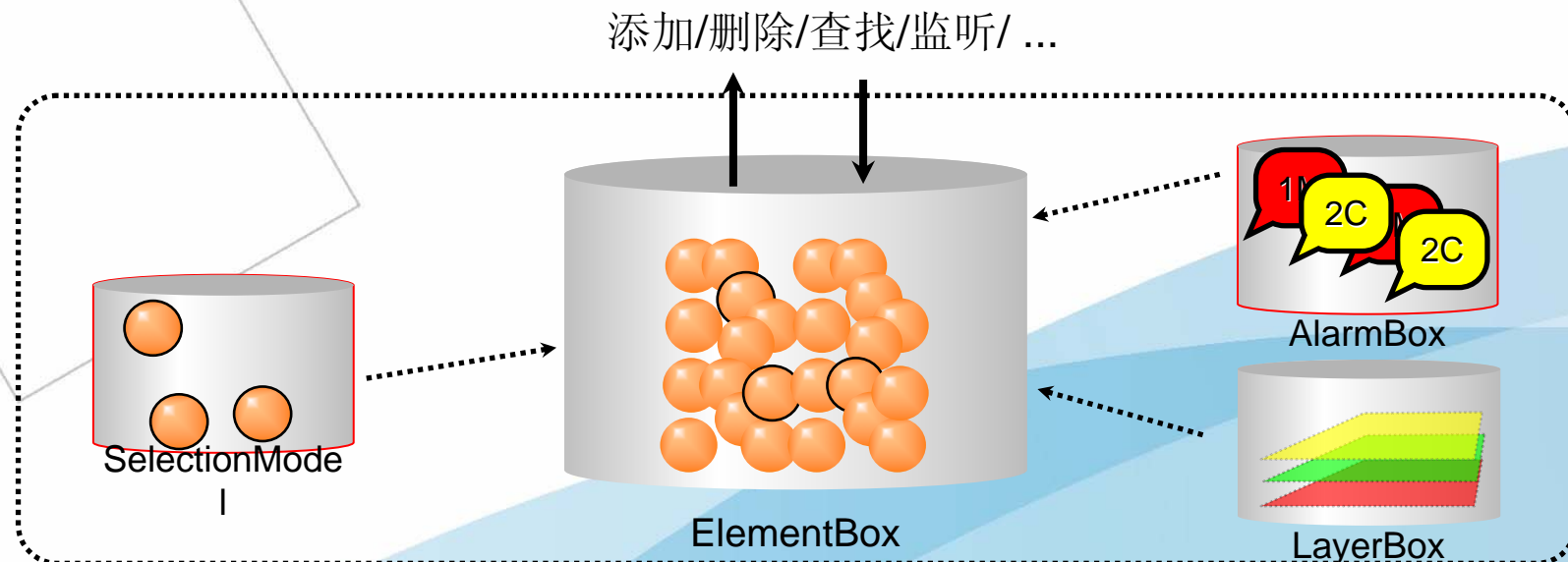
ElementBox的使用

ElementBox: 用于管理网元（IElement）的数据容器。
继承于DataBox，保留了DataBox的功能，增加了图层管理，告警管理

ElementBox#

```
public function get layerBox():LayerBox
```

```
public function get alarmBox():AlarmBox
```



ElementBox的使用

- 增加了layerBox、alarmBox

public function get layerBox():LayerBox

public function get alarmBox():AlarmBox

- 增加了按图层遍历

public function forEachByLayer(callbackFunction:Function, layer:ILayer = **null**):**void**

public function forEachByLayerReverse(callbackFunction:Function, layer:ILayer = **null**):**void**

- 增加了层次变化监听器

public function addIndexChangeListener(listener:Function, priority:int = 0, useWeakReference:Boolean = **false**):**void**

拓扑网元

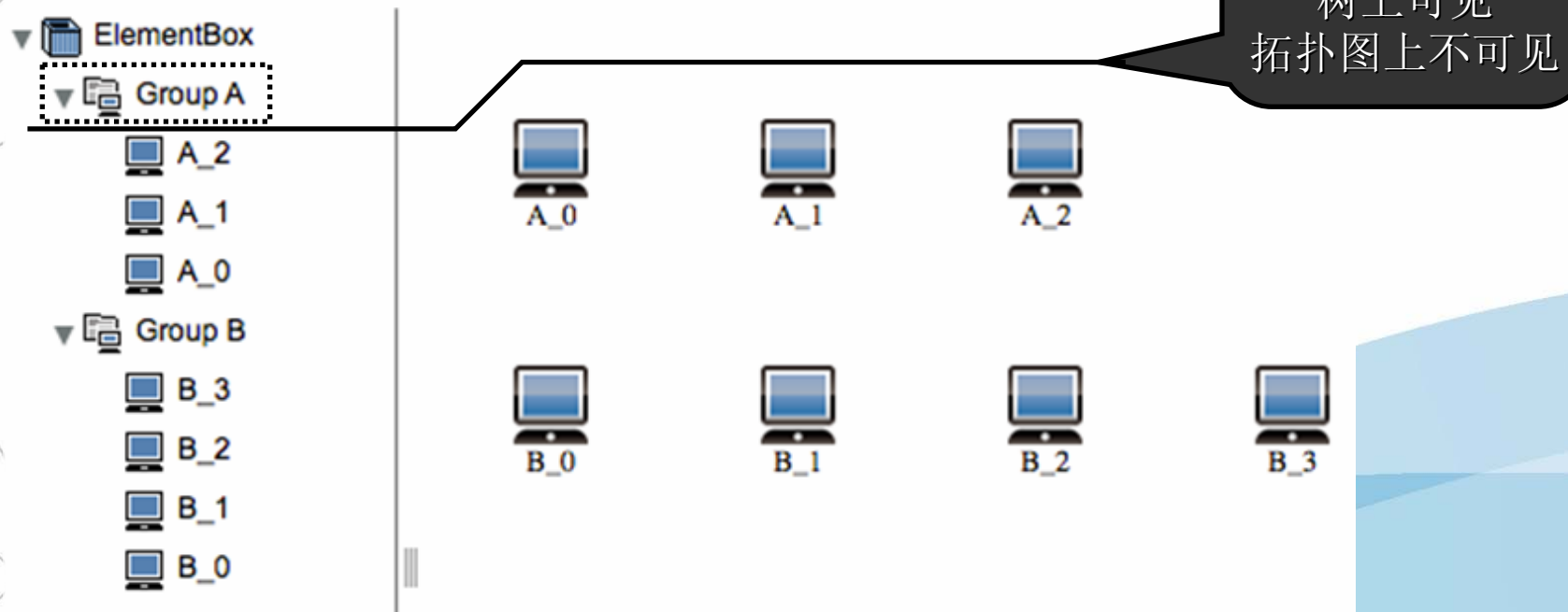
Data

- | -- **Alarm** - 告警元素
- | -- **Layer** - 图层元素
- | -- **Element** - 拓扑元素
 - | -- **Dummy** - 隐蔽节点
 - | -- **Node** - 节点
 - | -- **Follower** - 跟随者
 - | -- **Group** - 分组
 - | -- **Grid** - 网格
 - | -- **ShapeNode** - 多边形
 - | -- **Bus** - 总线
 - | -- **SubNetwork** - 子网
 - | -- **Link** - 连线
 - | -- **ShapeLink** - 折线
 - | -- **LinkSubNetwork** - 连线子网

Dummy

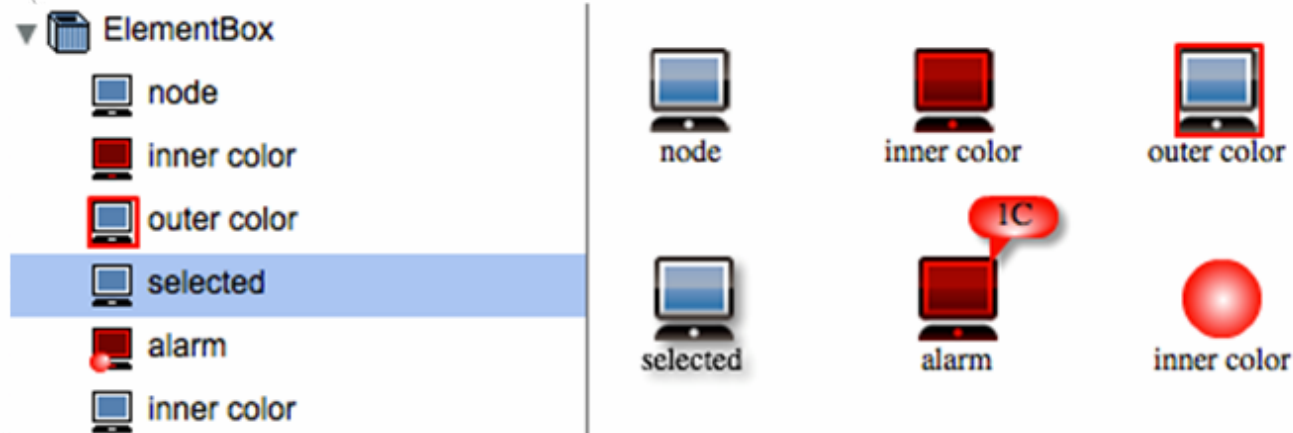
Dummy在Network中不可见，在树、表格中都可见

Dummy对象用来组织树结构，不影响Network



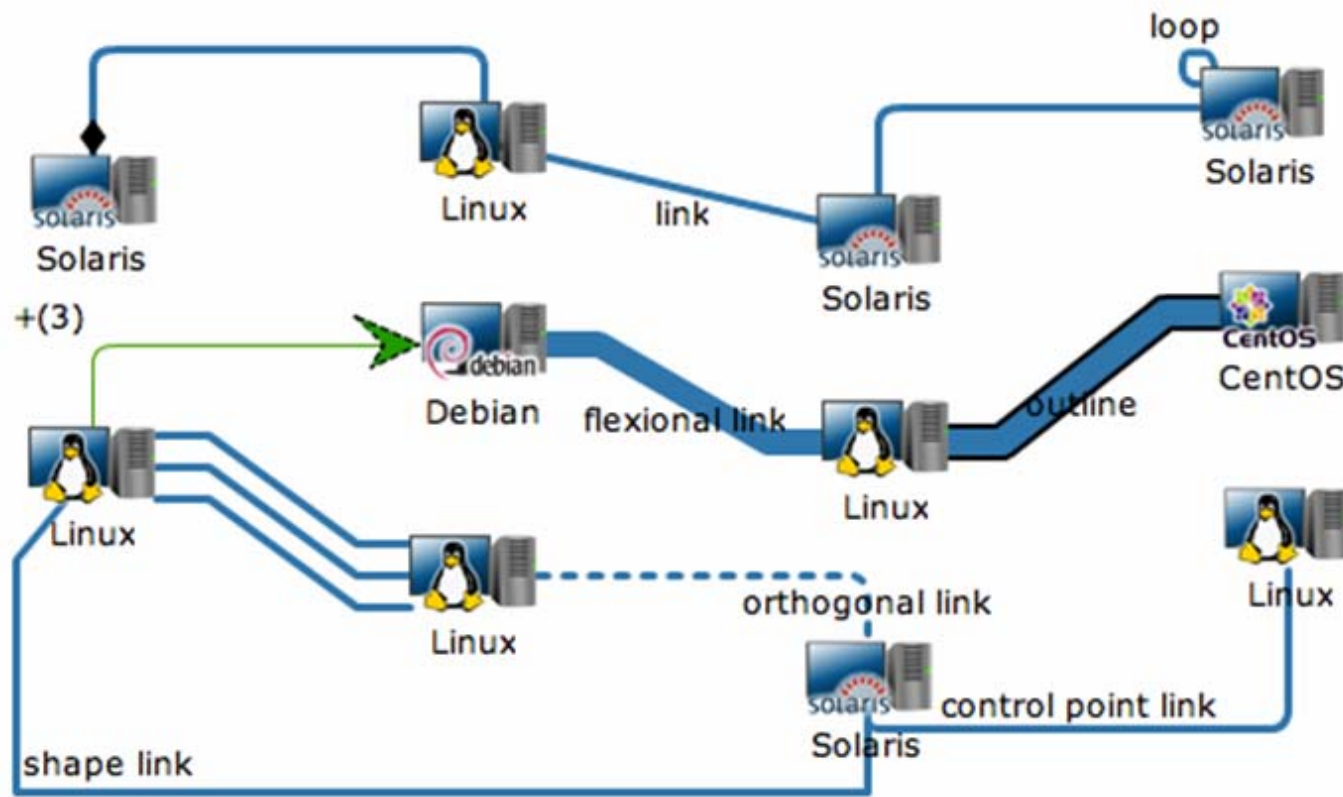
Node

- 普通网元，可以设置图片，边框，颜色渲染...
- twaver.Node 是其他主要网元类型的基类



Link

- 连线，连接节点的网元类型，可以设置多种样式，支持自环，绑定，箭头，虚线效果等

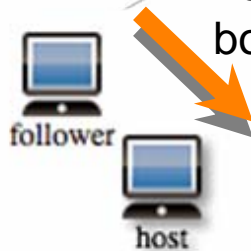


Follower

- 跟随者，可设置host节点，host移动，follower网元跟随移动

```
var node:Follower = new Follower();  
node.location = new Point(100, 100);  
node.name = 'follower';  
box.add(node);
```

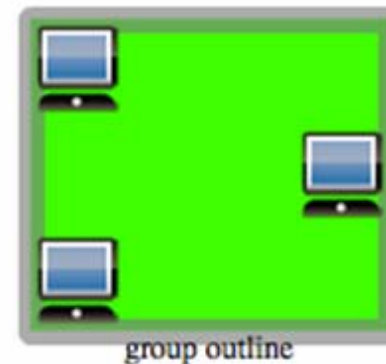
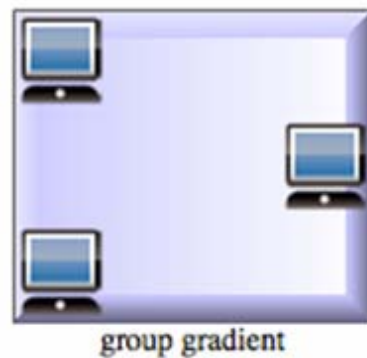
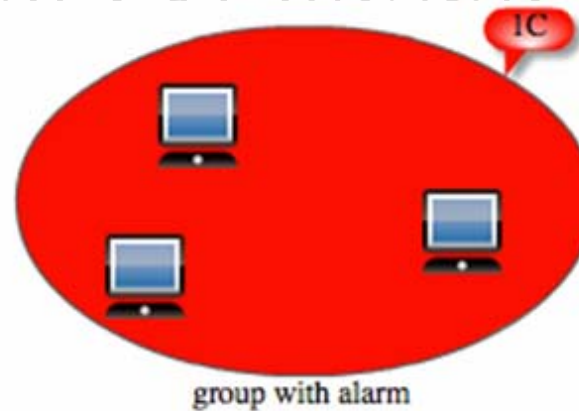
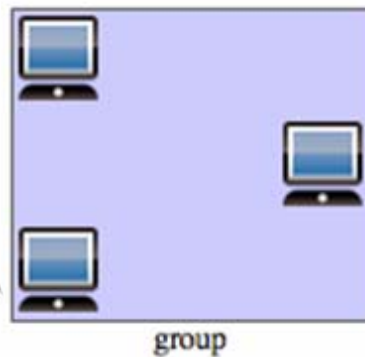
```
var host:Node = new Node();  
host.location = new Point(140, 140);  
node.host = host;  
host.name = 'host';  
box.add(host);
```



host 移动，follower也跟着移动

Group

- 分组，可以设置椭圆，矩形等形状，支持渐变填充，边框颜色等渲染效果



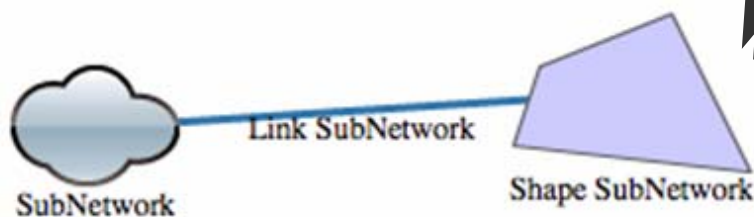
SubNetwork

- 子网，通过双击操作，可以进入或退出子网

进入子网，则显示该子网中的网元，退出子网，则进入上一级子网，当前子网为null时，则表示为顶层

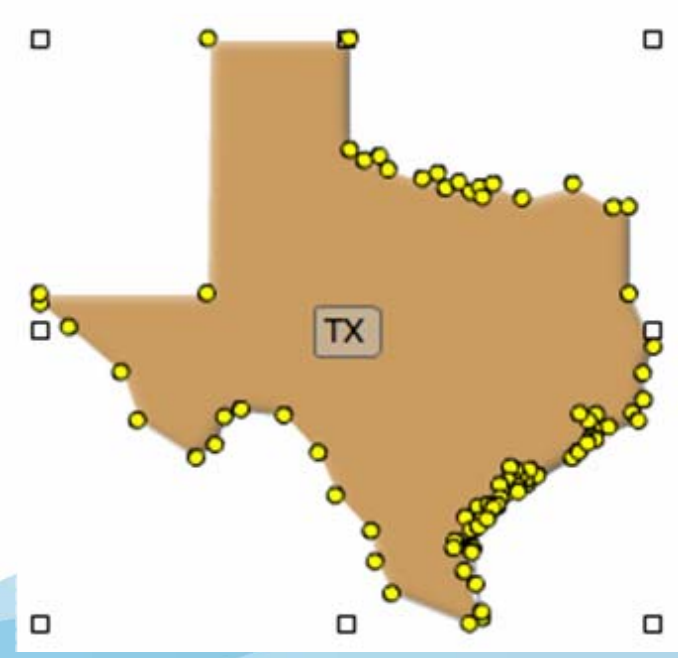
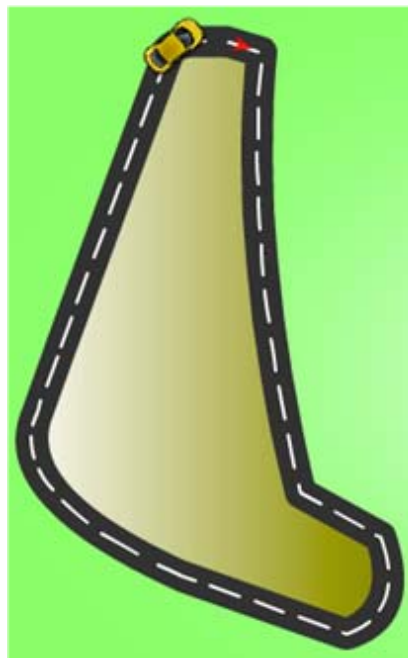
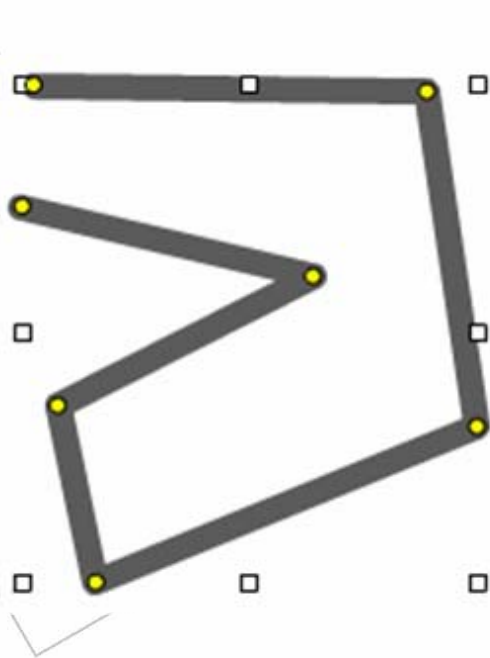
- 子网下可以设置自己的背景

都实现于ISubNetwork接口



ShapeNode

- 多边形，由一系列控制点围成的多边形或线段



Bus

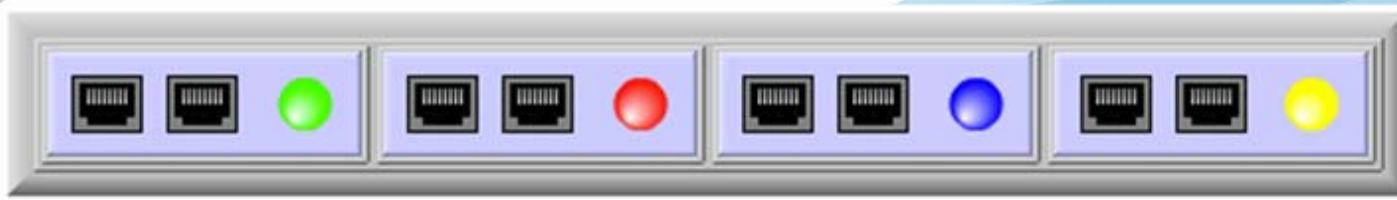
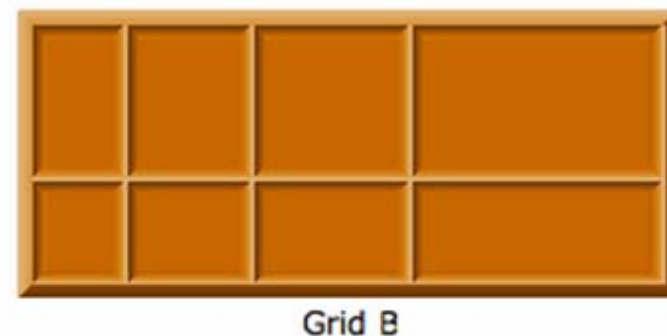
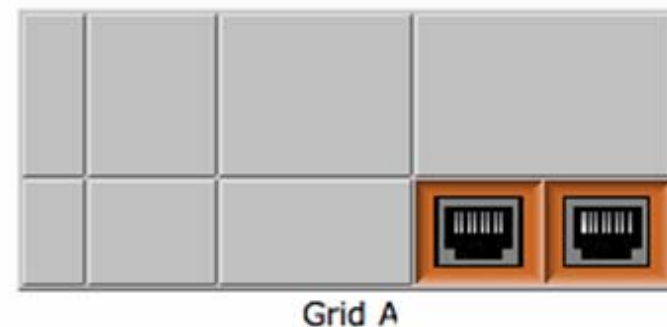
- 继承于ShapeNode，通过一组控制点围成的线段，与其相连的Link呈垂直链接分布



Grid

- 网格，行列分布，可以组合使用，制作设备面板图

```
var grid:Grid = new Grid();  
grid.name = 'Grid B';  
grid.setStyle(Styles.GRID_BORDER, 5);  
grid.setStyle(Styles.GRID_COLUMN_COUNT, 4);  
grid.setStyle(Styles.GRID_COLUMN_PERCENTS, [0.15,0.2,0.25,0.4]);  
grid.setStyle(Styles.GRID_ROW_COUNT, 2);  
grid.setStyle(Styles.GRID_ROW_PERCENTS, [0.6,0.4]);  
grid.setStyle(Styles.GRID_FILL_COLOR, 0xCC6600);  
grid.setStyle(Styles.GRID_DEEP, 11);  
grid.setStyle(Styles.GRID_CELL_DEEP, -4);  
grid.setStyle(Styles.GRID_PADDING, 0);  
grid.width = 250;  
grid.height = 106;
```



网元属性

twaver.Consts#

```
public static const PROPERTY_TYPE_ACCESSOR:String =  
"accessor"; public static const PROPERTY_TYPE_CLIENT:String =  
"client";  
public static const PROPERTY_TYPE_STYLE:String = "style";
```

Element

accessor - name, id ...

node.name = '001';

style - 样式属性

node.setStyle(Styles.INNER_COLOR, 0xFF0000);

client - 用户属性

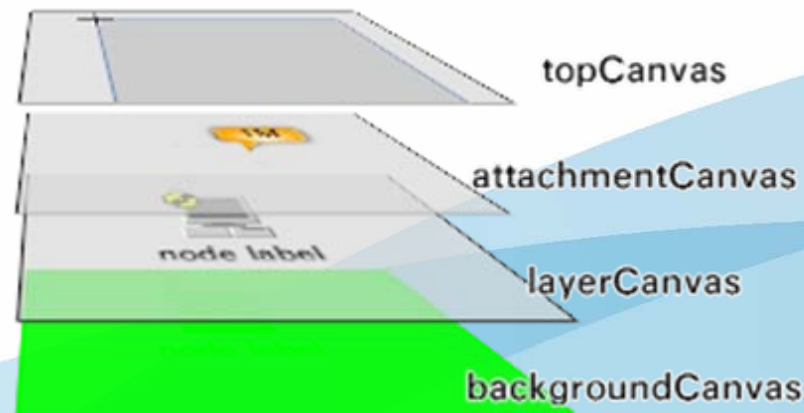
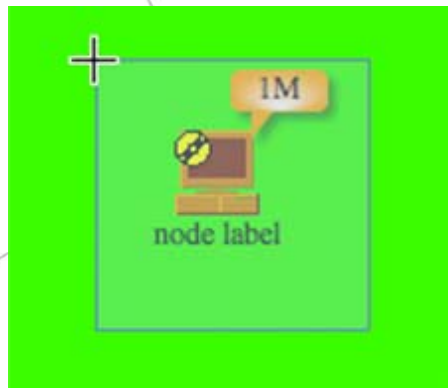
node.setClient('age', 27);

Network的使用

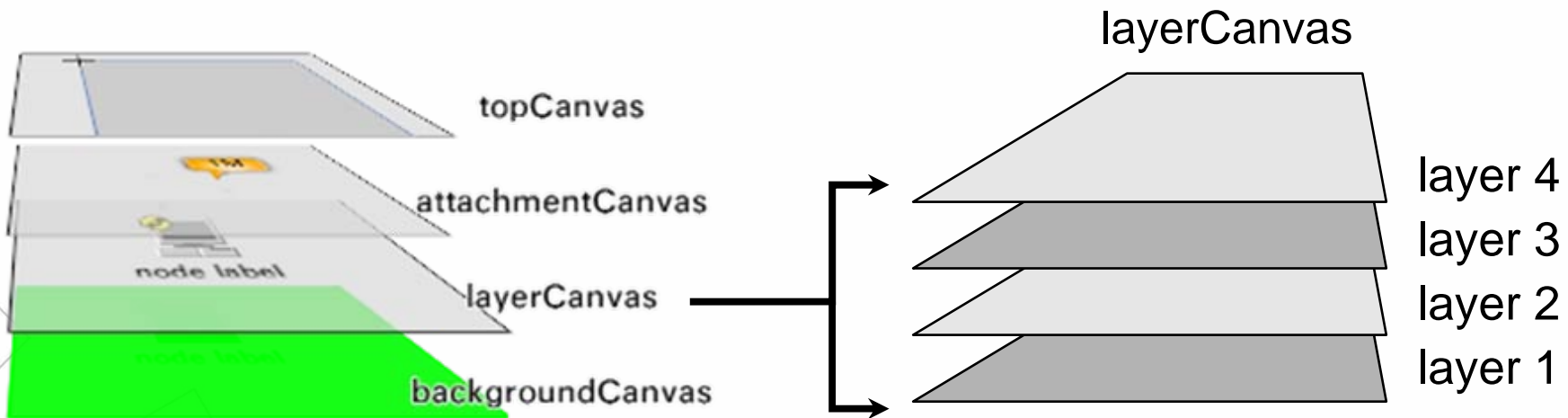
- Network的层次结构
- Network添加背景
- Network的交互处理
- Network的过滤器
- Network样式规则函数
- Network自动布局

Network的层次结构

- rootCanvas ——主画布，所有拓扑组件都在其中
- topCanvas ——顶层画布，可用于绘制交互框选框，调整大小时的拖拽块
- attachmentCanvas ——放置顶层附件组件
- layerCanvas ——所有网元视图所在层
 - layer n
 - layer ...
 - default layer
- bottomCanvas ——底层画布，可用于在背景图之上绘制底纹
- backgroundCanvas ——背景画布，包括颜色，图片等背景



图层管理



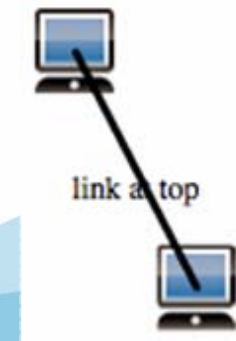
```

var box:ElementBox = network.elementBox;
var layerBox:LayerBox=box.layerBox;

var layerTop:Layer=new Layer("top", 'top');
layerBox.add(layerTop, layerBox.count);

var node:Node = new Node();
box.add(node);
var node2:Node = new Node();
box.add(node2);
var link:Link = new Link(node, node2);
link.layerID = layerTop.id;
link.name = 'link at top';
link.setStyle(Styles.LINK_COLOR, 0x000000);
box.add(link);

```



TM
Tmaver

添加背景

- 可以对**ElementBox**和**Subnetwork**设置背景，支持栅格图片，颜色，颜色渐变等；

```
box.setStyle(Styles.BACKGROUND_TYPE, Consts.BACKGROUND_TYPE_IMAGE);  
box.setStyle(Styles.BACKGROUND_IMAGE, "background");
```

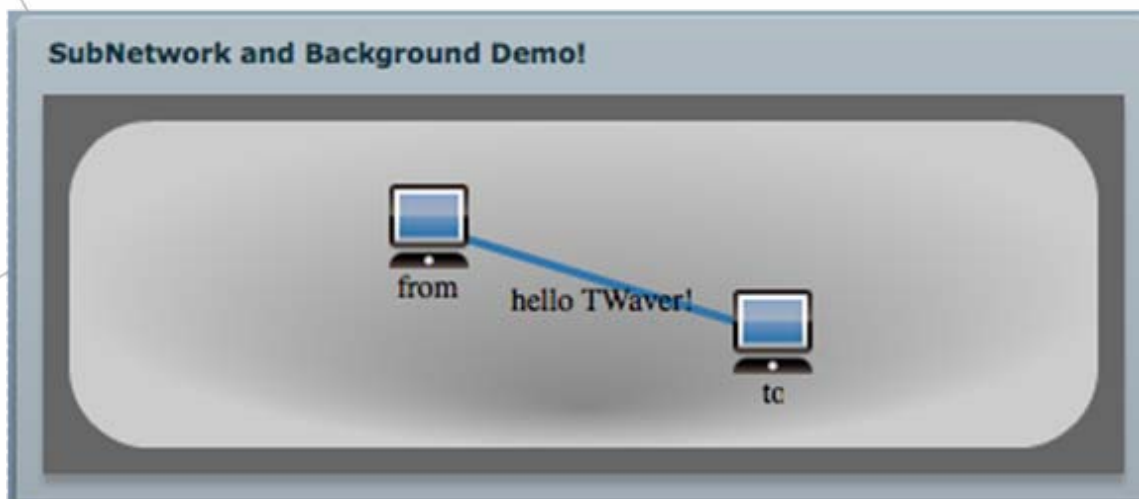
- 也可以使用**Flex**组件默认支持的背景样式，如：

```
<twaver:Network id="network" backgroundColor="0x666666" width="100%"  
height="100%" />
```

子网背景示例

```
<twaver:Network id="network" backgroundColor="0x666666" width="100%" height="100%" />
```

```
var subnetwork:SubNetwork=new SubNetwork();  
subnetwork.setStyle(Styles.BACKGROUND_TYPE, Consts.BACKGROUND_TYPE_VECTOR);  
subnetwork.setStyle(Styles.BACKGROUND_VECTOR_GRADIENT, Consts.GRADIENT_RADIAL_SOUTH);  
subnetwork.setStyle(Styles.BACKGROUND_VECTOR_FILL_COLOR, 0xCCCCCC);  
subnetwork.setStyle(Styles.BACKGROUND_VECTOR_GRADIENT_COLOR, 0x888888);  
subnetwork.setStyle(Styles.BACKGROUND_VECTOR_SHAPE, Consts.SHAPE_ROUNDRECT);  
subnetwork.setStyle(Styles.BACKGROUND_VECTOR_PADDING, 10);
```



Network交互模式

- **InteractionHandler** - 单个监听器
- **Interaction Mode** - 一组监听器组成一种交互模式

Network#

```
public function set interactionHandlers(interactionHandlers:ICollection):void
```

- **Network**预定义了多种交互模式，如默认模式，编辑模式，缩放模式...

Network#

```
public function setDefaultInteractionHandlers(lazyMode:Boolean = false):void
```

```
public function setEditInteractionHandlers(lazyMode:Boolean = false):void
```

```
public function setCreateLinkInteractionHandlers(linkClass:Class = null):void
```

...

交互定制示例

interaction handler 实现类的基本结构

```
public class HighlightInputHandler extends BasicInteractionHandler{  
    public function HighlightInputHandler(network:Network){  
        super(network);  
    }  
    override public function installListeners():void{  
        this.network.addEventListener(MouseEvent.CLICK, handleClick);  
    }  
    override public function uninstallListeners():void{  
        this.network.removeEventListener(MouseEvent.CLICK, handleClick);  
    }  
    ...  
}
```

设置拓扑图的交互模式

```
network.interactionHandlers = new Collection([  
    new SelectInteractionHandler(network),  
    new MoveInteractionHandler(network),  
    new DefaultInteractionHandler(network),  
    new HighlightInputHandler(network)  
]);
```

参考TWaver中文社区文章: 《TWaver Flex 拓扑图交互模式的定制》



鼠标划过，网元高亮

```

public class HighlightInputHandler extends BasicInteractionHandler{
    public function HighlightInputHandler(network:Network){
        super(network);
    }
    override public function installListeners():void{
        this.network.addEventListener(MouseEvent.MOUSE_MOVE, handleMouseMove);
    }
    override public function uninstallListeners():void{
        this.network.removeEventListener(MouseEvent.MOUSE_MOVE, handleMouseMove);
    }
    private function handleMouseMove(e:MouseEvent):void{
        var element:IElement = network.getElementByMouseEvent(e);
        if(element != null){
            highlight(element);
        }else{
            reset();
        }
    }
    private var highlightElement:IElement;
    private var highLightColor:uint = 0xFF8888;
    private var oldColor:int = -1;

    [Embed(source="hand_cursor.png")]
    private var handCursor:Class;

    private function highlight( element:IElement):void {
        if(element == null || element == highlightElement){
            return;
        }
        reset();
        highlightElement = element;
        oldColor = element.getStyle(Styles.INNER_COLOR);
        CursorManager.setCursor(handCursor, 2, -9, -3);
        element.setStyle(Styles.INNER_COLOR, highLightColor);
    }
    private function reset():void {
        if (highlightElement != null) {
            if(oldColor){
                highlightElement.setStyle(Styles.INNER_COLOR, oldColor);
            }else{
                highlightElement.setStyle(Styles.INNER_COLOR, null);
            }
            highlightElement = null;
            CursorManager.removeAllCursors();
        }
    }
}

```

TWaver™

Network过滤器

过滤器用于全局控制网元状态，例如控制网元的显示或隐藏，可移动还是不可移动

Network#

//可见过滤器

```
public function get/set visibleFunction():Function
```

//可移动过滤器

```
public function get/set movableFunction():Function
```

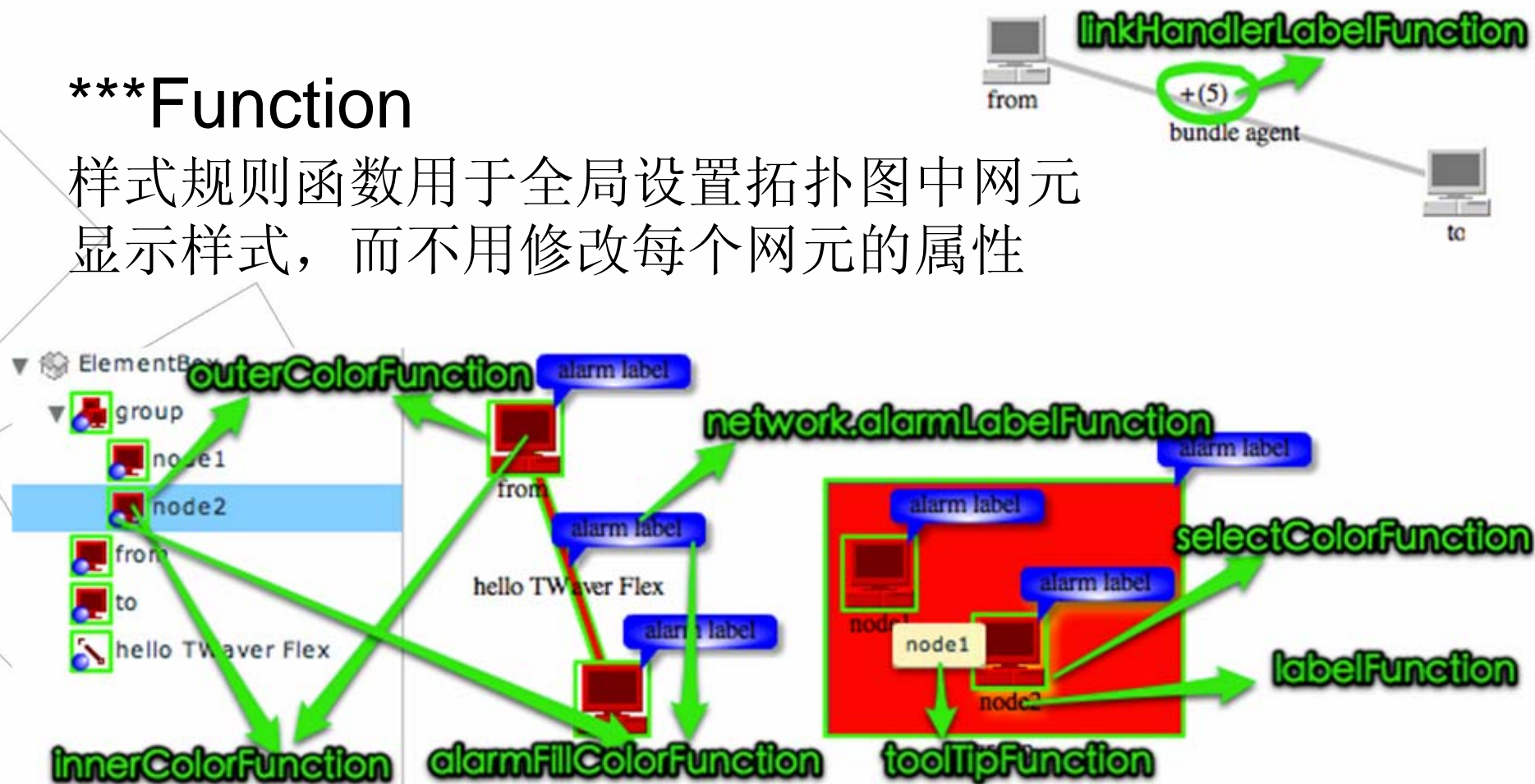
//可编辑过滤器

```
public function get/set editableFunction():Function
```

Network样式规则函数

***Function

样式规则函数用于全局设置拓扑图中网元显示样式，而不用修改每个网元的属性

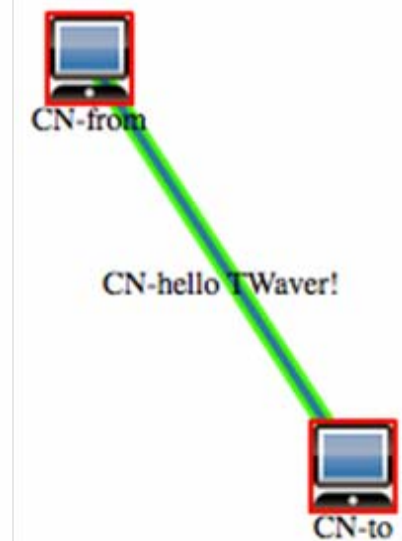


Network样式规则函数示例

对网元文本标签增加前缀，定制外边框颜色规则的示例：

```
network.labelFunction = function(element:IElement):String{  
    return "CN-" + element.name;  
};
```

```
network.outerColorFunction = function(element:IElement):uint{  
    return (element is Node) ? 0xFF0000 : 0x00FF00;  
};
```

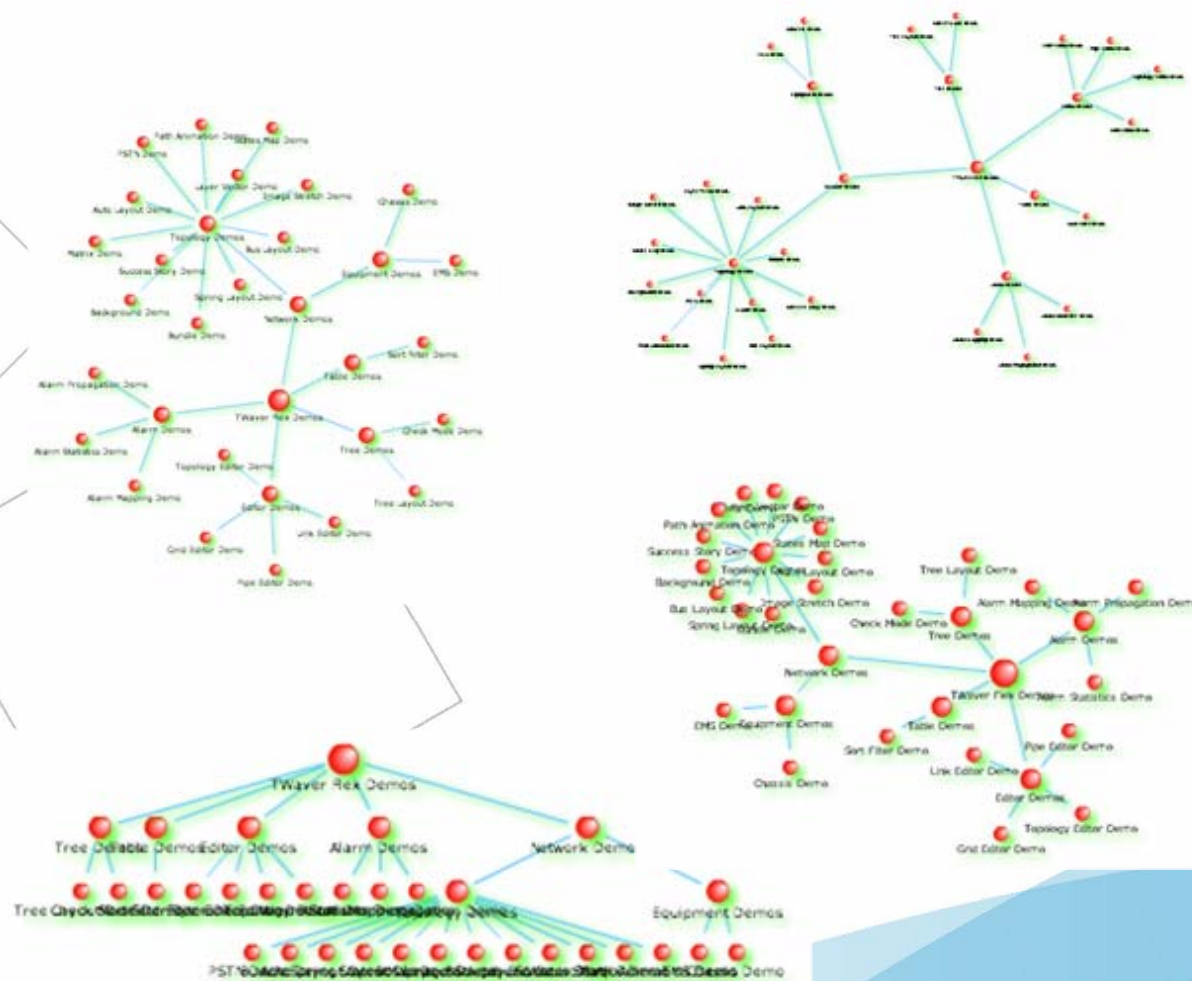


Network自动布局

- 自动对网元连线定位，使其分散开，分布呈现好的效果



默认自动布局





- 论坛: twaver.servasoft.com/forum
- MSN: twavercn@hotmail.com
- 邮件列表: twaver-news@servasoft.com

TWaver Flex 培训课程—— TWaver Flex通用组件的使用

Serva Software LLC

通用组件和告警的使用

●通用组件的使用

- Tree的使用
- Table的使用
- Chart的使用

●告警的使用

- 告警对象 / 告警级别 / 告警容器 / 告警状态 / 告警统计 / 告警呈现

●Flex组件扩展

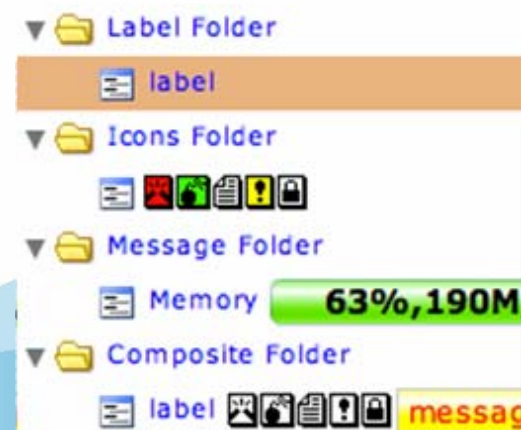
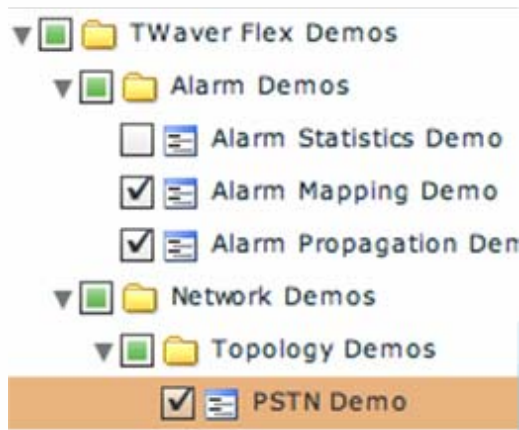
- CSS配置 / 定制皮肤 / 继承组件类

Tree的使用

twaver.controls.Tree 用于展示DataBox中元素父子层次关系的组件

```
var tree:Tree = new Tree(box);
```

```
<twaver:Tree id="tree" width="100%" height="100%" />
```



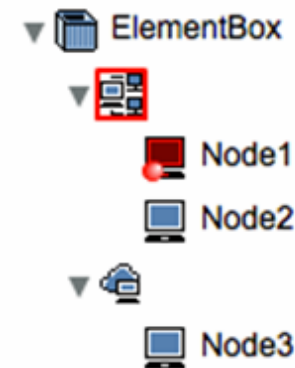
Tree的创建

```
var box:DataBox = new ElementBox();  
tree.dataBox = box;
```

```
var group:Group = new Group();  
box.add(group);  
var subnetwork:SubNetwork = new SubNetwork();  
box.add(subnetwork);
```

```
var node1:Node = new Node();  
node1.name = "Node1";  
node1.parent = group;  
node1.alarmState.increaseNewAlarm(AlarmSeverity.CRITICAL);  
box.add(node1);  
var node2:Node = new Node();  
node2.name = "Node2";  
node2.parent = group;  
box.add(node2);  
var node3:Node = new Node();  
node3.name = "Node3";  
node3.parent = subnetwork;  
box.add(node3);
```

```
tree.callLater(function():void{  
    tree.expandAll();  
});
```



网元的父子关系决定树的层次关系

定制树节点

- 设置图标

`data.icon = 'iconName';`

- 设置节点标签

`data.name = '001';`

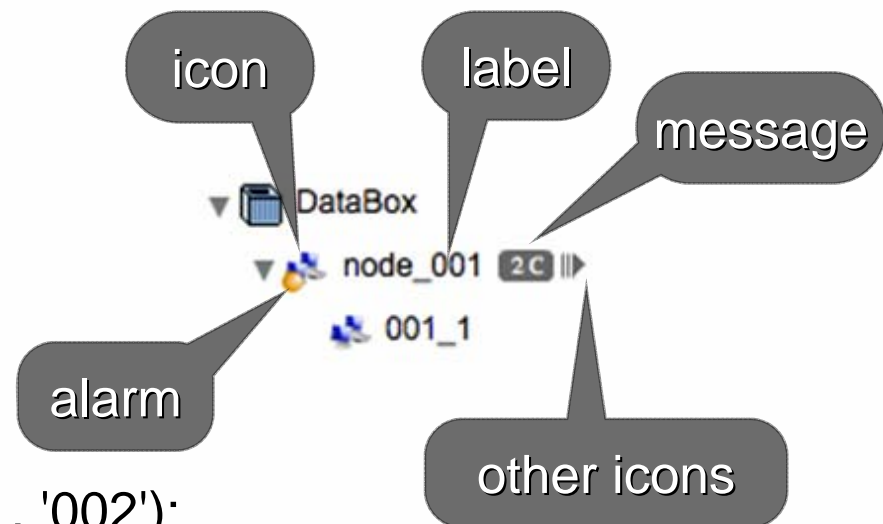
`element.setStyle(Styles.TREE_LABEL, '002');`

- 其他树节点样式

`element.setStyle(Styles.TREE_LABEL_***, ***);`

`element.setStyle(Style.TREE_MESSAGE_***, ***);`

`element.setStyle(Style.TREE_ICONS_***, ***);`



树节点样式示例

```
[Embed(source="images/device.png")]
private static const deviceIcon:Class;
[Embed(source="images/out.gif")]
private static const outIcon:Class;
private function init():void{
    Utils.registerImageByClass("deviceIcon",deviceIcon);
    Utils.registerImageByClass("outIcon",outIcon);

    var box:DataBox = tree.dataBox;

    var node:Element = new Element();
    node.name = '001';
    node.icon = 'deviceIcon';
    node.setStyle(Styles.TREE_LABEL, 'node_' + node.name);
    node.setStyle(Styles.TREE_ICONS_NAMES, ['outIcon']);
    node.setStyle(Styles.TREE_LAYOUT_GAP, 3);
    node.setStyle(Styles.TREE_MESSAGE, '2 C');
    node.setStyle(Styles.TREE_MESSAGE_FILL_COLOR, 0x666666);
    node.setStyle(Styles.TREE_MESSAGE_GRADIENT, Consts.GRADIENT_NONE);
    node.setStyle(Styles.TREE_MESSAGE_SIZE, 9);
    node.setStyle(Styles.TREE_MESSAGE_YPADDING, -3);
    node.setStyle(Styles.TREE_MESSAGE_COLOR, 0xFFFFFFFF);
    node.setStyle(Styles.TREE_LAYOUT, Consts.TREE_LAYOUT_LABEL_MESSAGE_ICONS);
    node.setStyle(Styles.TREE_ALARM_FILL_COLOR, 0xE08000);
    box.add(node);
}
```



树节点层次与顺序

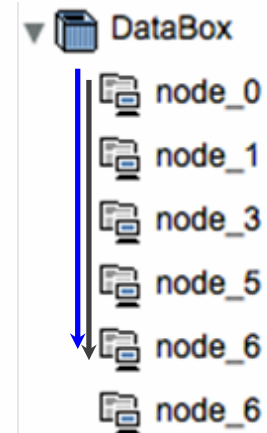
- 层次按网元父子关系
`node.parent = parent;`
`node.addChild(child);`
- 同层节点顺序
默认按加入到**DataBox**的先后顺序
可以通过调整网元在**dataBox**中的顺序来更改树节点的次序
`box.move**(node);`
- 可以设置比较器进行排序
`tree.compareFunction = compareFunction;`

树节点排序示例

```
var box:DataBox = tree.dataBox;
```

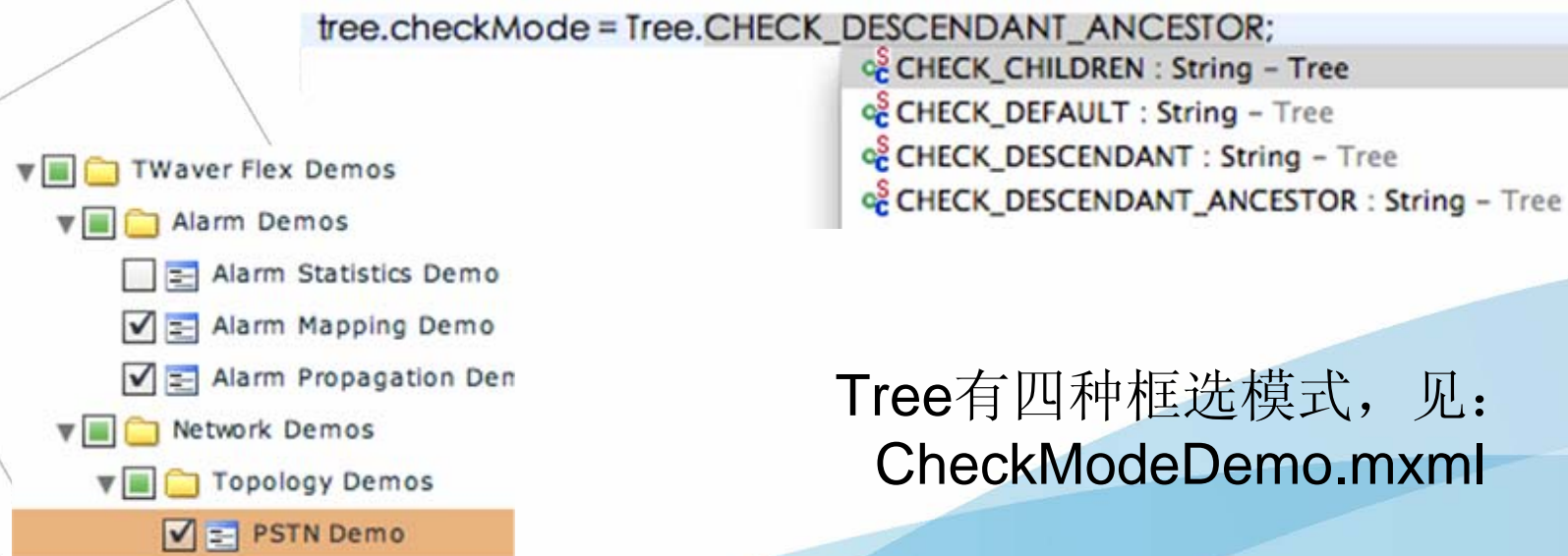
```
for(var i:int = 0; i < 10; i++){  
    var data:Data = new Data();  
    data.name = 'node_' + Utils.randomInt(10);  
    box.add(data);  
}
```

```
tree.compareFunction = function(d1:IData, d2:IData):int{  
    return d1.name.localeCompare(d2.name);  
};
```



Tree的框选模式

- Tree提供框选模式，可通过下面的方法切换
`Tree.checkMode = Tree.CHECK_`***



Tree有四种框选模式，见：
[CheckModeDemo.mxml](#)

Table的使用

- 与DataBox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
- 支持数据元素的三类属性的显示：
accessor、style、client
- 数据属性变化，同步更新，可编辑
- 继承于mx.controls.DataGrid，支持
renderer、editor定制

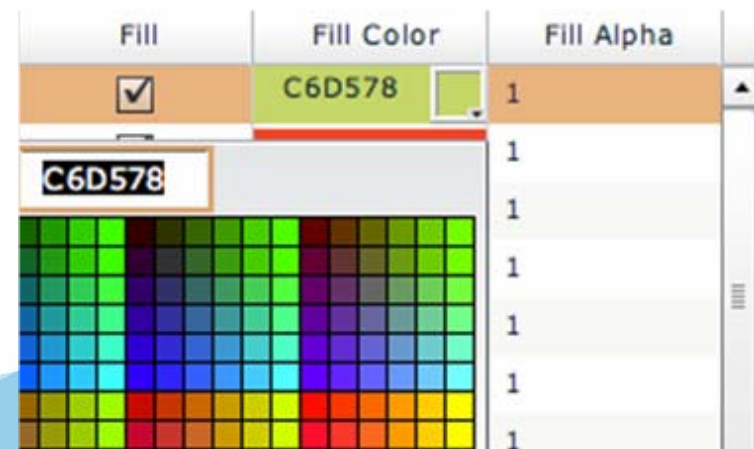
Table的使用

`twaver.controls.Table` 用于展示DataBox中元素属性的组件，
每行对应一个数据元素

```
var table:Table = new Table(box);
```

```
<twaver:Table id="table" width="100%" height="100%" />
```

Name ▼	Number	Inner Color
node_5	5	16229533
node_4	4	5731022
node_3	3	4504295
node_2	2	4997537
node_1	1	8846834



Table示例——mxml方式

支持三类属性：
accessor、style、client

```
<twaver:Table id="table1" editable="true">
  <twaver:columns>
    <twaver:TableColumn dataField="name" headerText="Name"/>
    <twaver:TableColumn dataField="C:number" headerText="Number"
      itemEditor="{new ClassFactory(mx.controls.NumericStepper)}"
      editorDataField="value"/>
    <twaver:TableColumn dataField="S:{Styles INNER_COLOR}"
      headerText="Inner Color" editable="false" />
  </twaver:columns>
</twaver:Table>
var box:DataBox = table1.dataBox;
var i:int=0;
while(i++<5){
  var node:Node = new Node();
  node.name = "node_"+i;
  node.setClient("number",i);
  node.setStyle(Styles.INNER_COLOR, Utils.randomColor);
  box.add(node);
}
```

Name ▼	Number	Inner Color
node_5	5	16229533
node_4	4	5731022
node_3	3	4504295
node_2	2	4997537
node_1	1	8846834

Table示例——API方式

```
var table2:Table = new Table(box);  
table2.editable = true;
```

```
var nameColumn:TableColumn = new TableColumn("Name");  
nameColumn.dataField = 'name';
```

```
var clientColumn:TableColumn = new TableColumn("Number");  
clientColumn.client = 'number';  
clientColumn.itemEditor = new ClassFactory(mx.controls.NumericStepper);  
clientColumn.editorDataField = 'value';
```

```
var colorColumn:TableColumn = new TableColumn("Inner Color");  
colorColumn.style = Styles.INNER_COLOR;  
colorColumn.editable = false;
```

```
table2.columns = [nameColumn, clientColumn, colorColumn];
```

```
table1.parent.addChild(table2);
```

表格数据顺序

- **Table**中数据存在三种顺序（正序，反序，根层）
 - `table.iterateMode = Consts. ITERATE_MODE_***;`
 - `Consts.ITERATE_MODE_REVERSE`
 - `Consts.ITERATE_MODE_FORWARD`
 - `Consts.ITERATE_MODE_ROOTS`
- 通过`dataBox.move***(element)`调整行的顺序

表格列排序

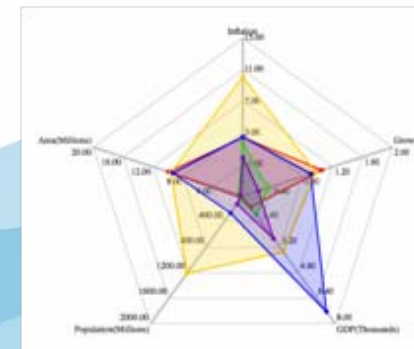
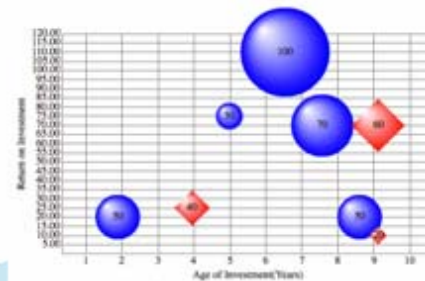
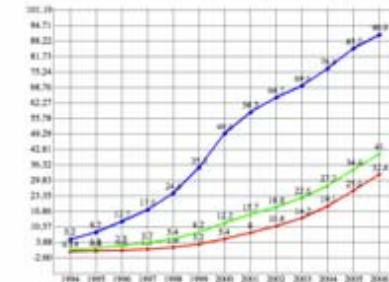
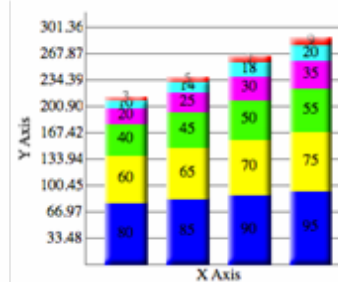
```
var sort:Sort = new Sort();  
//sort.fields = [new SortField("name"), new SortField("C:number")];  
sort.fields = [new SortField("name", false, true)];  
(table1.dataProvider as ArrayCollection).sort = sort;  
(table1.dataProvider as ArrayCollection).refresh();
```

Name ▼	Number	Inner Color
node_5	5	16229533
node_4	4	5731022
node_3	3	4504295
node_2	2	4997537
node_1	1	8846834

Name	Number	Inner Color
node_0	3	1923848
node_0	6	14986603
node_0	9	10550708
node_1	1	1426661
node_1	4	7830315
node_1	7	9846957

Chart的使用

- BarChart
- LineChart
- PieChart
- DialChart
- BubbleChart
- RadarChart



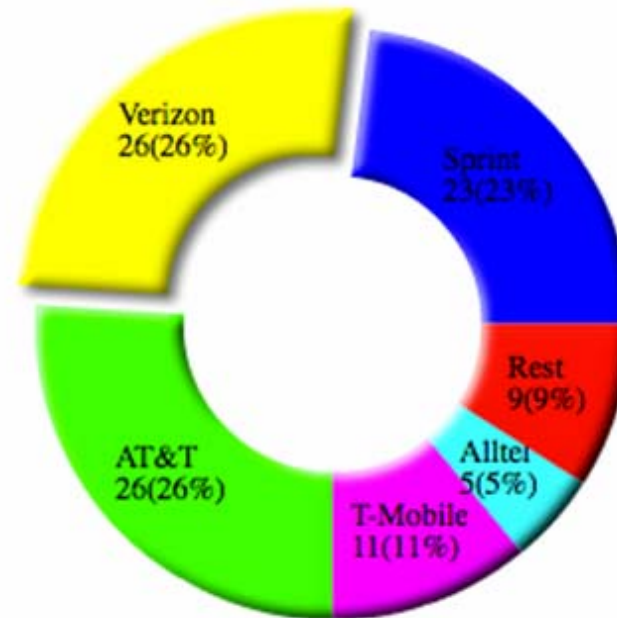
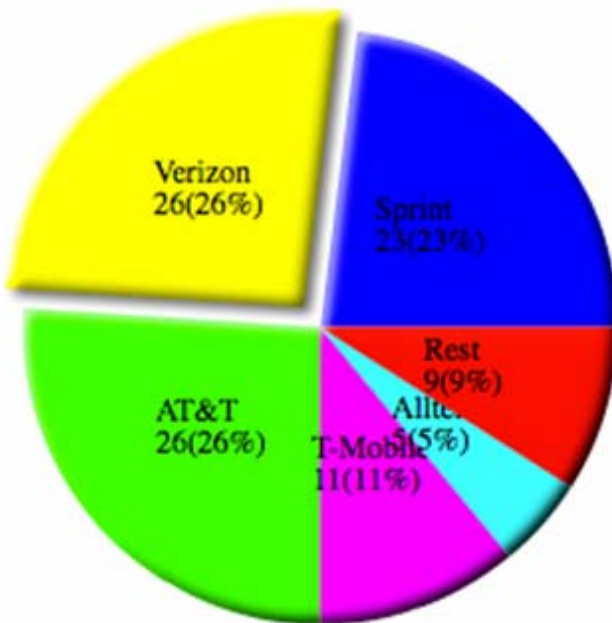
ChartValue & ChartValues

- 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
- Chart数值属性有两种：CHART_VALUE和CHART_VALUES，如下设置

```
element.setStyle(Styles.CHART_VALUE, 27);
```

```
element.setStyle(Styles.CHART_VALUES, [27, 37, 48]);
```

PieChart



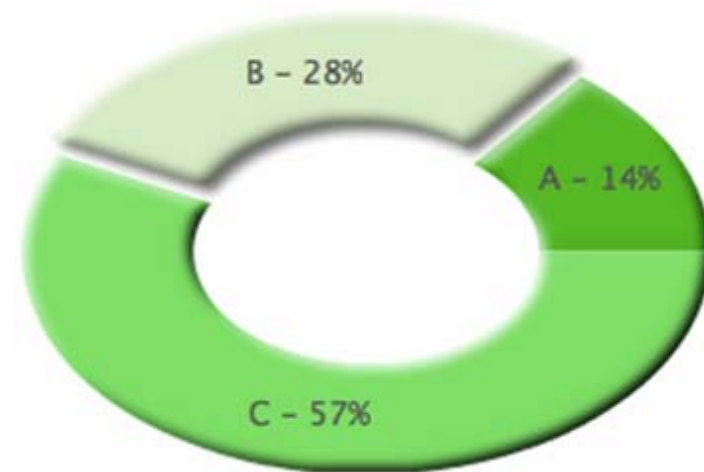
PieChart

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:twaver="http://www.servasoftware.com/2009/twaver/flex" >
    <mx:initialize>
        <![CDATA[
            import twaver.*;
            import twaver.network.Network;

            var box:DataBox = chart.dataBox;
            createData('A', 10, 0x4CB743);
            createData('B', 20, 0xD6EAC9);
            createData('C', 40, 0x77DD77);

            chart.type = Consts.PIECHART_TYPE_OVALDONUT;
            chart.valueTextFunction = function(item:IElement, value:Number):String{
                return item.name + ' - ' + int(value/chart.sum*100) + '%';
            };

            function createData(name:String, value:Number, color:uint):void{
                var data:IElement = new Element();
                data.setStyle(Styles.CHART_VALUE, value);
                data.setStyle(Styles.CHART_COLOR, color);
                data.setStyle(Styles.CHART_VALUE_COLOR, 0x555555);
                data.setStyle(Styles.CHART_VALUE_FONT, 'hevetica');
                data.name = name;
                box.add(data);
            };
        ]]>
    </mx:initialize>
    <twaver:PieChart id="chart" backgroundColor="0xFFFFF" width="100%" height="100%"/>
</mx:Application>
```



Twaver™

LineChart

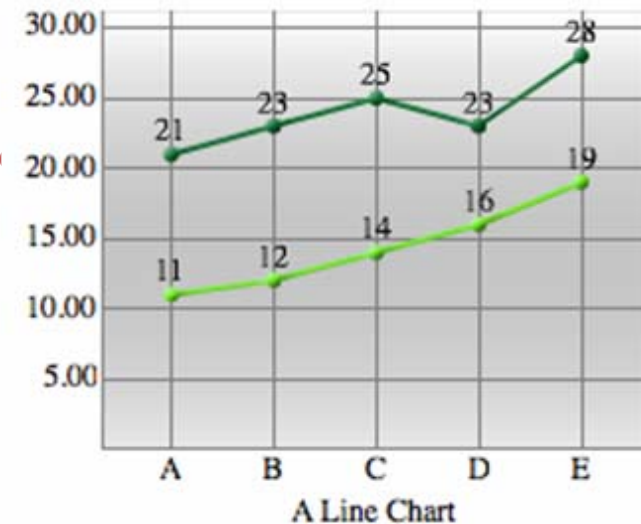
```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:twaver="http://www.servasoftware.com/2009/twaver/fl"
    <mx:initialize>
        <![CDATA[
            import twaver.*;
            import twaver.network.Network;

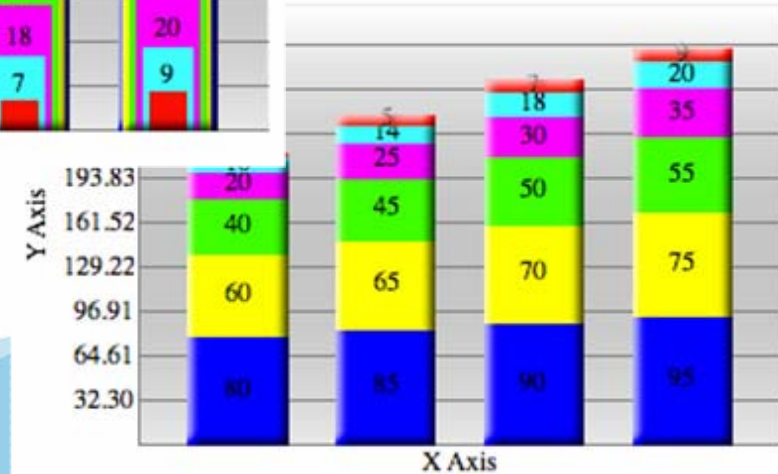
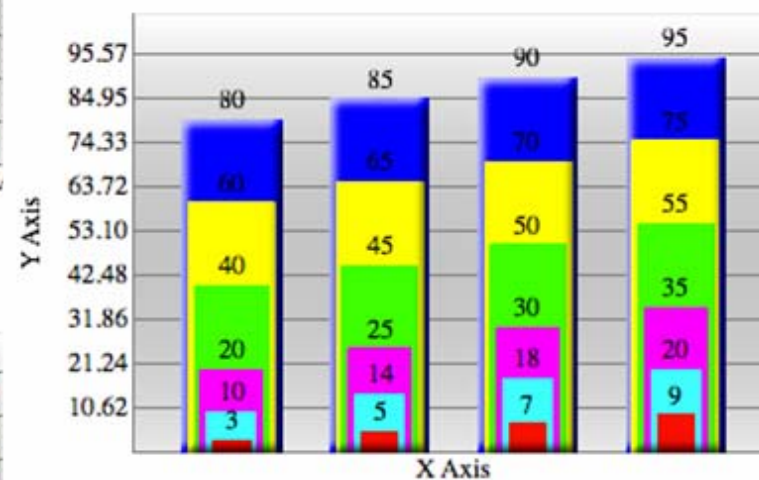
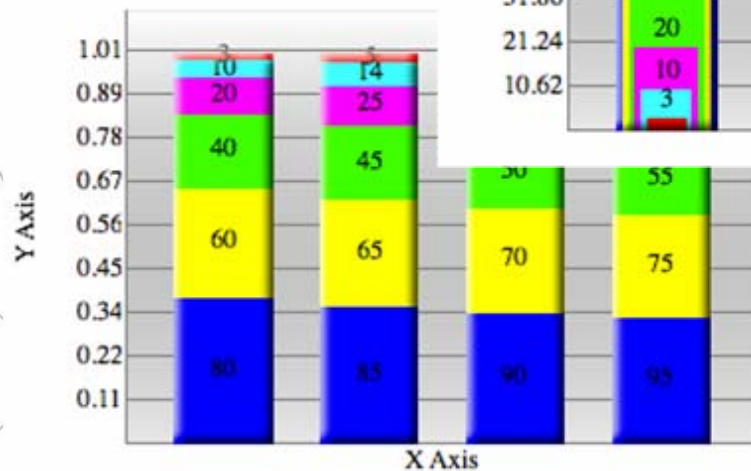
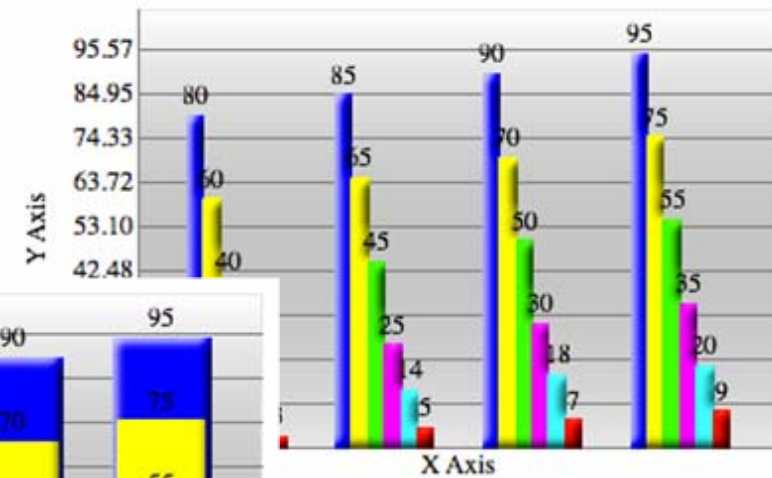
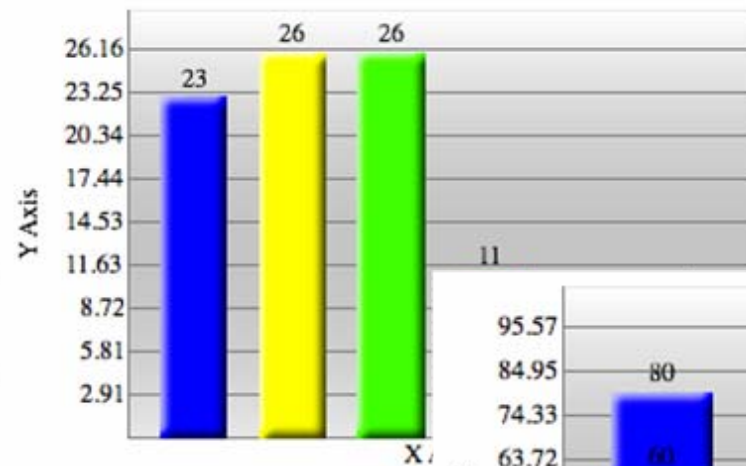
            var box:DataBox = chart.dataBox;
            var data:IElement = new Element();
            data.setStyle(Styles.CHART_VALUES, [11, 12, 14, 16, 19]);
            box.add(data);
            data = new Element();
            data.setStyle(Styles.CHART_VALUES, [21, 23, 25, 23, 28]);
            box.add(data);

            chart.xScaleTexts = ['A', 'B', 'C', 'D', 'E'];
            chart.lowerLimit = 0;
            chart.yScaleValueGap = 5;
            chart.xAxisText = 'A Line Chart';
            chart.backgroundVisible = true;
        ]]>
    </mx:initialize>
    <twaver:LineChart id="chart" backgroundColor="0xFFFFFFFF" width="100%" height="100%"/>
</mx:Application>

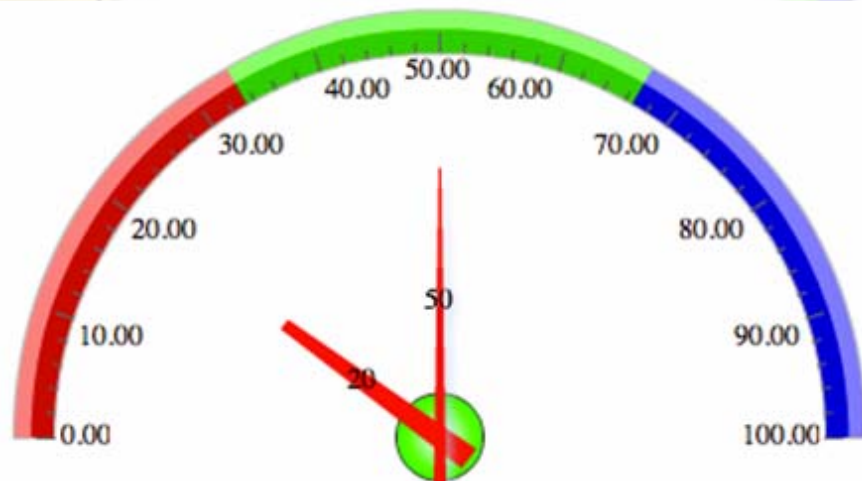
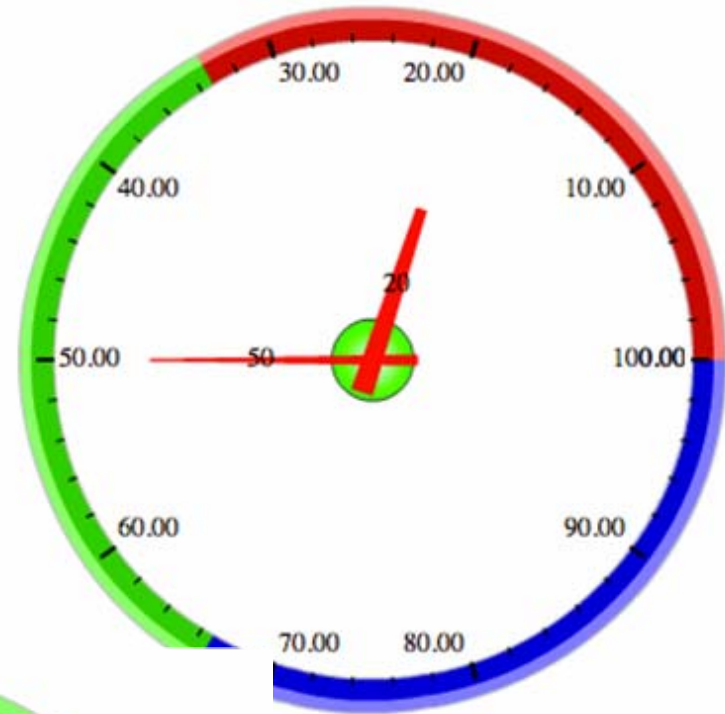
```



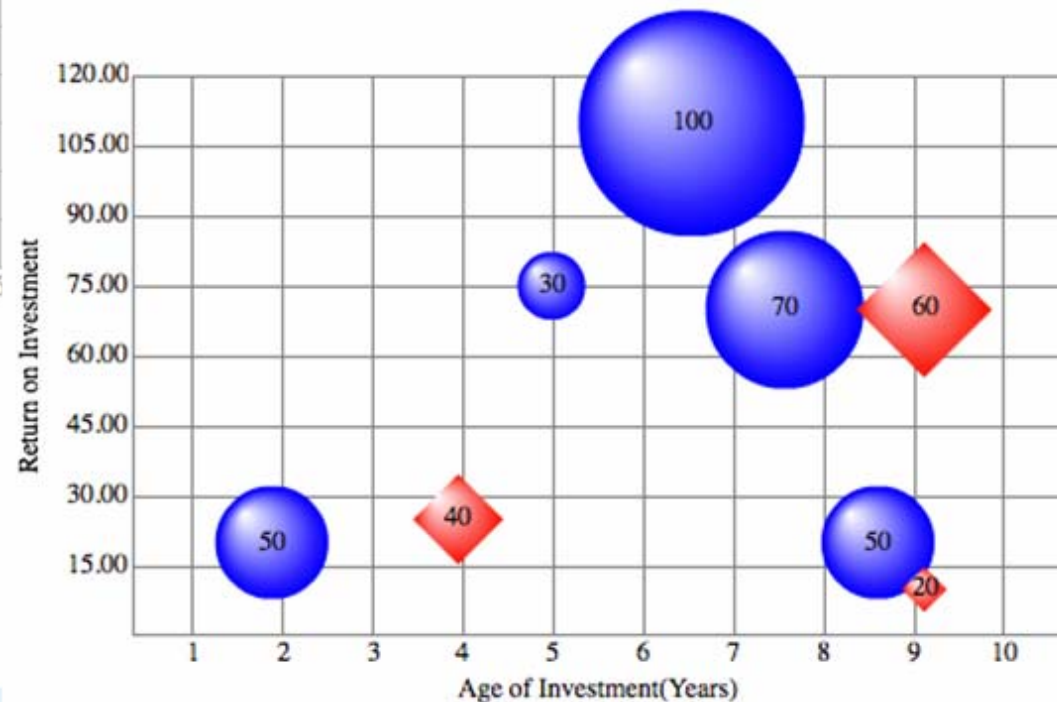
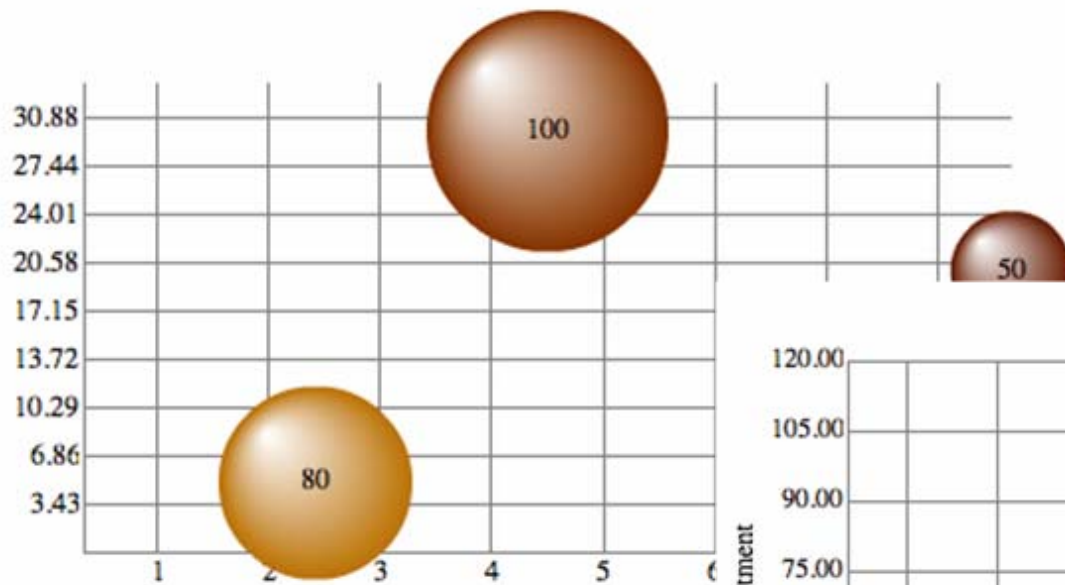
BarChart



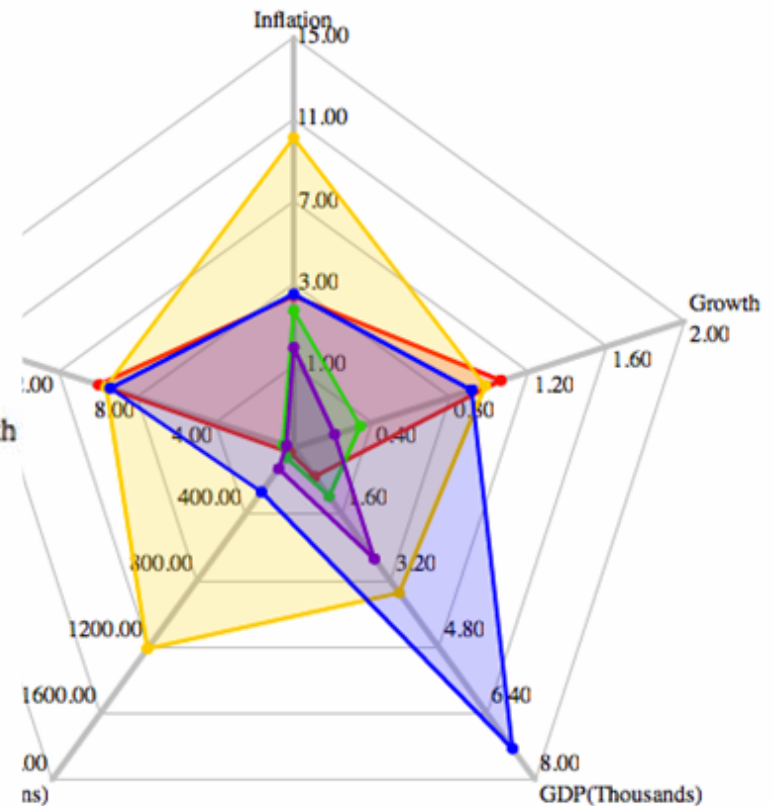
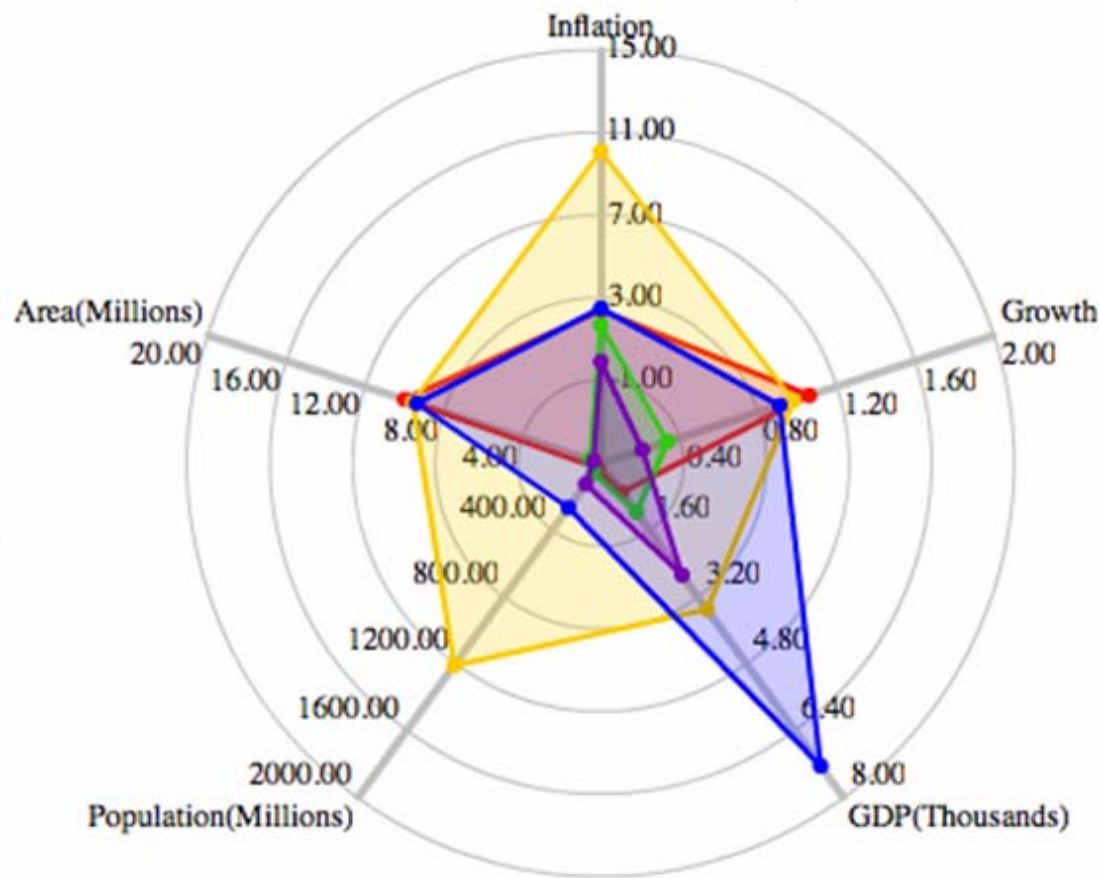
DialChart



BubbleChart



RadarChart



TM
waver

告警的使用

- 告警：反映网元或设备运行状态的业务元素
- 告警对象 / 告警级别
- 告警容器
- 告警状态 / 告警统计
- 告警呈现

告警级别

- twaver. AlarmSeverity
- 告警级别：反映告警的紧急程度
包含：name, nickName, value, color等属性
- 默认提供六级告警级别
CRITICAL / MAJOR / MINOR / WARNING /
INDETERMINATE / CLEARED
- 可以扩展告警级别
`AlarmSeverity.clear();`
`AlarmSeverity.register(1, "a", "a", 0xFF0000, "AAA");`

定制告警级别示例

```
private function init():void{
    AlarmSeverity.clear();

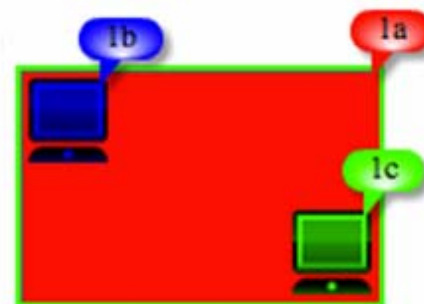
    var a:AlarmSeverity = AlarmSeverity.register(1, "a", "a", 0xFF0000, "AAA");
    var b:AlarmSeverity = AlarmSeverity.register(2, "b", "b", 0x0000FF, "BBB");
    var c:AlarmSeverity = AlarmSeverity.register(3, "c", "c", 0x00FF00, "CCC");

    var box:ElementBox = network.elementBox;

    var node1:Group = new Group();
    node1.expanded = true;
    var node2:Node = new Node();
    node2.setLocation(100, 100);
    var node3:Node = new Node();
    node3.setLocation(200, 150);
    node3.parent = node1;
    node2.parent = node1;
    box.add(node1);
    box.add(node2);
    box.add(node3);

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

private function addAlarm(alarmBox:AlarmBox, element:Element, severity:AlarmSeverity):void{
    var alarm:Alarm=new Alarm(null, element.id, severity);
    alarmBox.add(alarm);
}
```



告警对象

- **twaver.Alarm:** 描述设备故障或者网络异常的数据元素
- 每个告警对象中包含：
id, elementID、alarmSeverity、acked、cleared等
alarm.setClient(key, value)添加属性

告警容器

- AlarmBox: 管理告警对象的容器
- 提供增加删除清空告警函数
add/remove/clear ...
- 快速查找器 / 监听器...



AlarmBox

告警状态

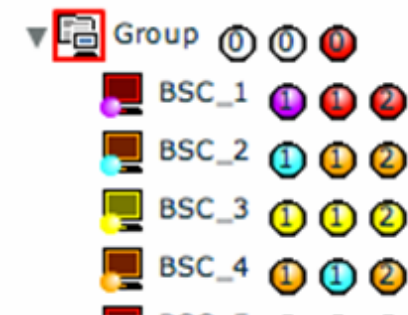
- **twaver.AlarmState:** 描述网元或设备当前的告警状态
- 包含网元的新发、确认、传递告警的统计信息
- 不包含具体的告警（**twaver.Alarm**）对象

告警状态统计

- **twaver.AlarmStateStatistics**告警状态统计：
对整个**elementBox**中所有网元告警状态进行统计
- 包含各种级别告警的数量
- 包含新发、确认告警的数量
- 支持过滤器，可指定对哪些网元做统计

告警的呈现

- 告警冒泡
- 告警渲染（染色，边框）
- 告警表格
- 告警统计图表



Mapping ID	Alarm Severity	acked	cleared	R
5	Indeterminate	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Fri Jun 17 1
4	Warning	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Fri Jun 17 1
3	Minor	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Fri Jun 17 1
2	Major	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Fri Jun 17 1

Severity	New	Acked	Total ▲
Cleared	2	1	3
Minor	1	3	4
Critical	3	2	5
Major	5	1	6
Indeterminate	0	7	7
Warning	8	1	9
Total	19	15	34

Flex组件扩展

- Flex组件扩展主要有三种途径：
 - CSS设置
 - 定制皮肤
 - 继承组件类

CSS配置

```
@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/mx";
```

```
global{
  chrome-color: #FFC000;
}
#button1{
  fontSize: 22;
}
.big{
  fontSize: 18;
}
s|Button{
  fontSize: 14;
}
mx|Button{
  fontSize: 11;
}
```

全局

#ID

.StyleName

命名空间|类名

命名空间申明

mx button, with 'mx|Button' style

spark button, with 's|Button' style

with '.big' style

with '#button1' style

```
<mx:Button label="mx button, with 'mx|Button' style" />
<s:Button label="spark button, with 's|Button' style" />
<s:Button styleName="big" label="with '.big' style" />
<s:Button id="button1" label="with '#button1' style" />
```

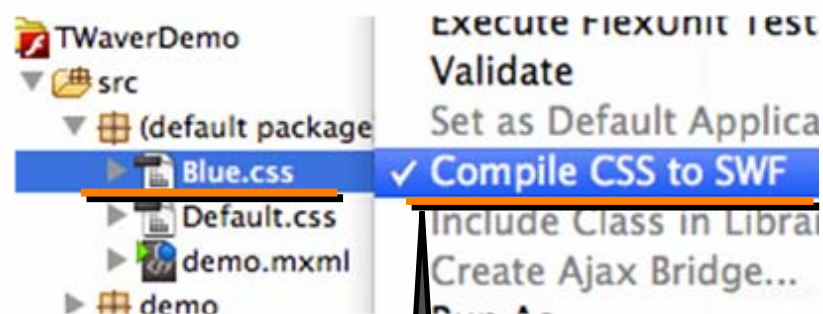
动态加载CSS

● 动态加载CSS

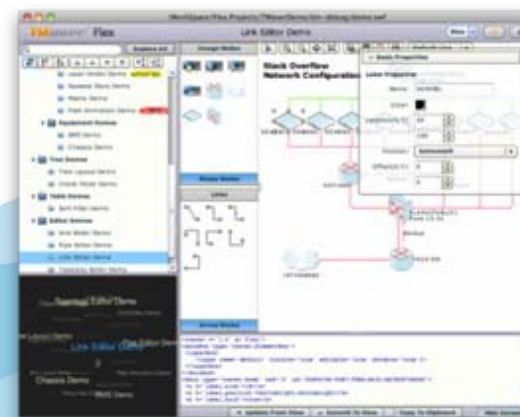
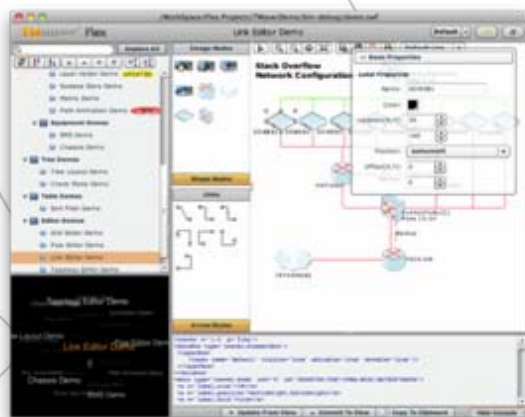
- `StyleManager.loadStyleDeclarations("Blue.swf", true);`

● 动态卸载CSS

- `StyleManager.unloadStyleDeclarations("Blue.swf", true);`

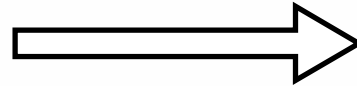


勾选“Compile CSS to SWF”选项



定制皮肤

默认文本框



定制皮肤的文本框



组件

```
public class CustomTextInput extends TextInput{  
    public function CustomTextInput(){  
        super();  
        //...  
        this.setStyle("skinClass", CustomTextInputSkin);  
    }  
}
```

皮肤

```
<s:SparkSkin ...>...  
    <s:HGroup>  
        <!-- icon -->  
        <s:Image ... />  
        <!-- text -->  
        <s:RichEditableText ... />  
    </s:HGroup>  
</s:SparkSkin>
```

通过设置'skinClass'样式，实现与定制皮肤的关联

定制皮肤示例

CustomTextInput.as

```
package component{
    import skin.CustomTextInputSkin;
    import spark.components.TextInput;

    [Style(name="icon", inherit="no", type="Class")]
    [Style(name="radius", inherit="true", type="Number")]

    public class CustomTextInput extends TextInput{
        [Embed(source="/images/search.png")]
        private const defaultIcon:Class;

        public function CustomTextInput(){
            super();
            this.setStyle('icon', defaultIcon);
            this.setStyle("skinClass", CustomTextInputSkin);
        }
    }
}
```

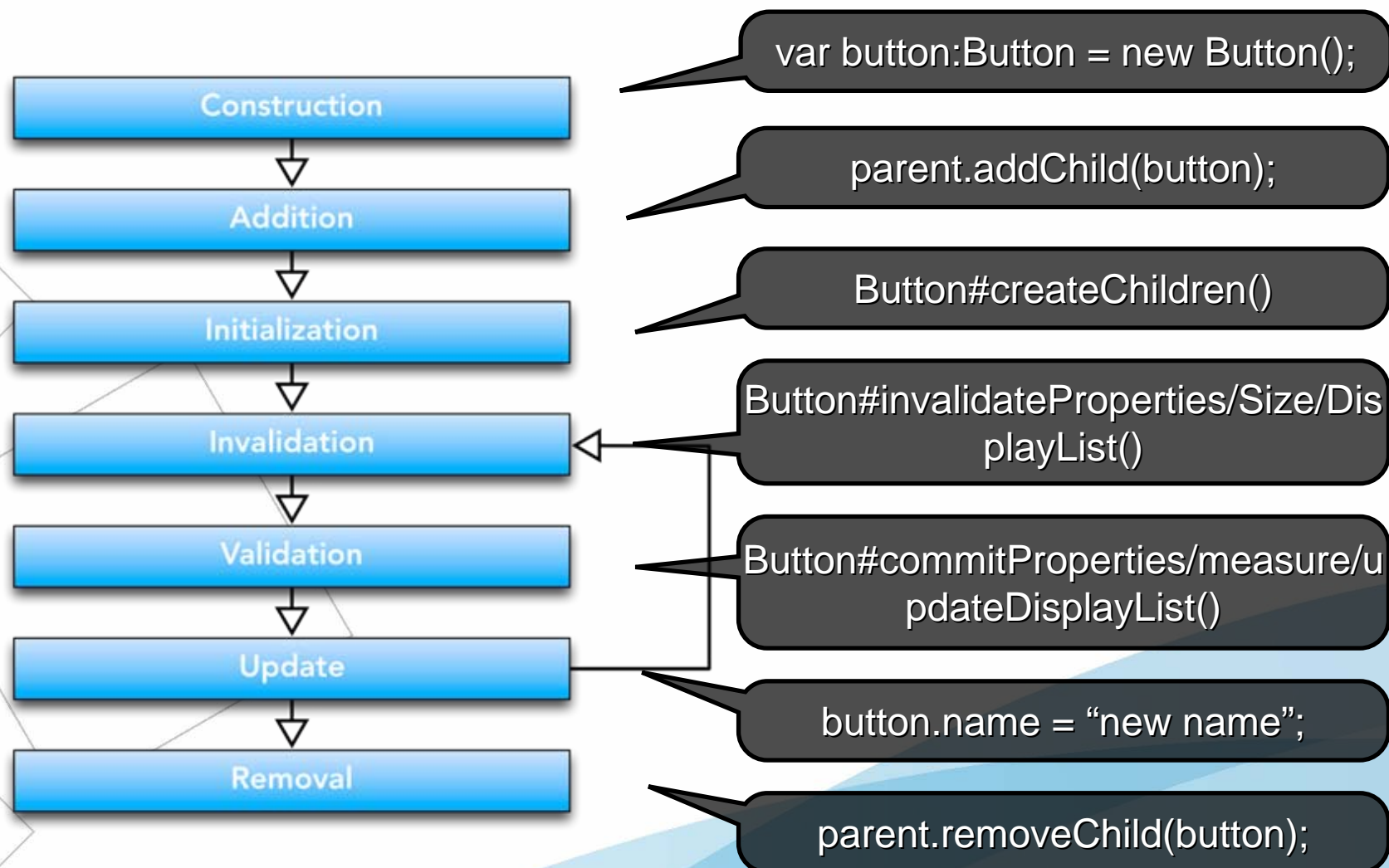
CustomTextInputSkin.mx

```
<s:SparkSkin ...> ml
    ...
    <s:HGroup gap="0" height="100%" paddingLeft="4"
paddingRight="4">
        <!-- icon -->
        <s:Image id="icon" includeIn="normal" x="0"
y="0"
source="{hostComponent.getStyle('icon')}"
verticalAlign="middle" height="100%"/>
        <!-- text -->
        <s:RichEditableText id="textDisplay"
            verticalAlign="middle"
            widthInChars="10"
            left="1" right="1" top="1" bottom="1"
height="100%"/>
    </s:HGroup>
    ...
</s:SparkSkin>
```

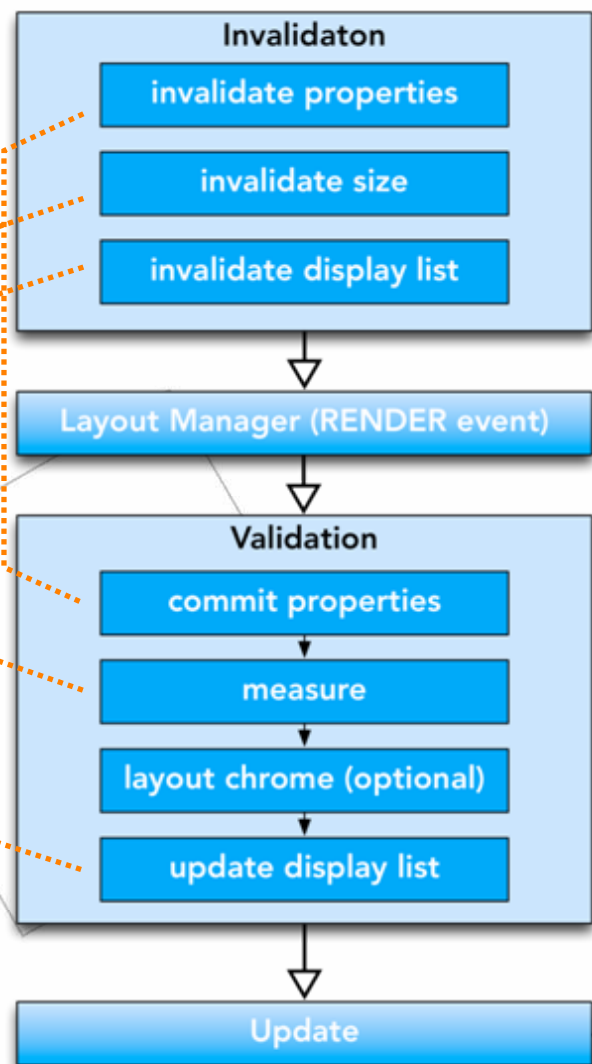
继承组件类

- **Flex**组件的基本类是**UIComponent**
- 定制组件，通常需要重写**UIComponent**的**createChildren()/commitProperties()/measure()/updateDisplayList()**等方法
- 深入扩展组件需要了解**Flex**组件的生命周期和无效-生效循环

Flex组件生命周期



无效-生效循环



三个函数分别对应属性变化，大小变化，显示变化
三者可以单独调用，也可以组合调用

组件无效后，不会马上生效，需等到
RENDER事件中执行生效

对应于无效的三种类型，外加
layoutChrome()
三者依次调用

组件变化，重复“无效-生效循环”

Thank you

- 论坛: twaver.servasoft.com/forum
- Email: tw-service@servasoft.com
- 邮件列表: twaver-news@servasoft.com