生成日期:

第一次改动:第二次改动:

TreeView 类

父类

Control

简介

带有根节点和子节点的树形数据展示.

详细说明

这是一个Control类, 以多级父节点与子节点的形式, 依照用户设置的模板ItemFactory显示用户通过DataModel设置的数据. ItemFactory与DataModel必须由用户设置, 且需要满足一定的条件. 见ItemFactory与DataModel小节.

布局上的Horizontal ScrollBar和Vertical ScrollBar可以通过方法替换.

布局说明

滚动条部分:

和ItemView不同, TreeView会渲染所有被展开的节点, 不管是不是在界面上的可见范围内. 这样布局的结构比ItemView简单很多.

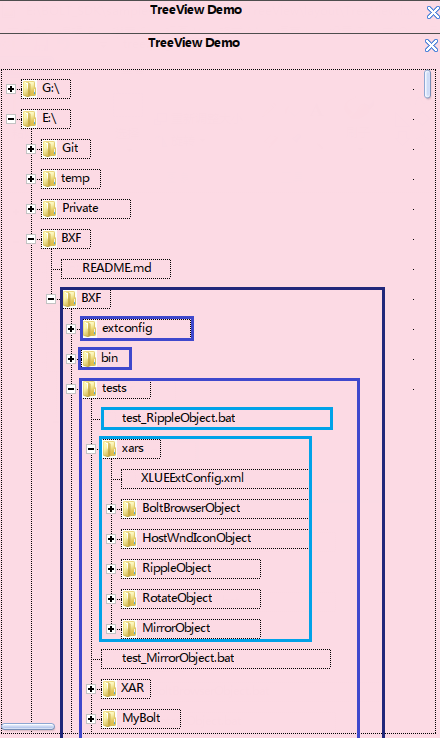
在TreeView Control内的根控件是ScrollPanel, 这是一个带有两个滚动条和一个内部对象的控件. 滚动条HScrollBar和VScrollBar会控制内部对象的位置. 内部对象的大小会影响滚动条的滚动范围.

TreeView会把包含所有节点对象的layout传给scroll panel作为内部对象来显示.

父节点, 子节点部分:

对每一个节点, TreeView会创建一个节点对象Node和一个layout对象, Node对象本身和所有子节点的layout都是这个layout的子对象. 下图的BXF节点处于layout的左上角, 除了显示BXF的虚线框之外, extconfig, bin, tests三个layout都是BXF所处的layout的children. 其中extconfig和bin节点因为没有展开, 它们的layout和它们的节点对象是一样大小和位置的, tests节点展开了, 所以tests节点的layout还包含其他对象.

总结一下就是:父节点的layout是子节点的layout的父控件.



成员方法列表

|  |  |
| --- | --- |
| 名称 | 简介 |
| RenderTreeFromDataTable() | 显示table存储的数据树 |
| SetStyle() | 设置样式 |
| SetItemFactory() | 设置创建节点对象的Item Factory |
| SetTreeModel() | 设置数据树的DataModel |
| SetHScrollBar() | 设置水平滚动条 |
| SetVScrollBar() | 设置垂直滚动条 |

属性列表

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 描述 |
| style\_name | String | WindowsStyle | 样式名称 |
| vertical\_indent | int | 0 | 垂直方向父子节点相对缩进 |
| horizontal\_indent | int | 30 | 水平方向父子节点相对缩进 (1) |
| icon\_width | int | 50 | 图标宽度 (2) |
| icon\_height | int | 20 | 图标高度 (3) |
| node\_width | int | 200 | 节点对象宽度 (4) |
| node\_height | int | 20 | 节点对象高度 (5) |
| vertical\_padding | int | 50 | 垂直方向同级节点间距 (6) |
| horizontal\_padding | int | 50 | 水平方向同级节点间距 (7) |
| joint\_open\_icon | string | nil | 展开节点图标 |
| joint\_close\_icon | string | nil | 收拢节点图标 |
| joint\_width | int | 15 | 展开/收拢节点图标宽度 (8) |
| joint\_height | int | 15 | 展开/收拢节点图标高度 |
| joint\_node\_padding | int | 0 | 展开/收拢节点图标和节点间距 |
| knob\_size | int | 5 | 节点到展开/收拢图标之间的间距 |

事件列表

|  |  |
| --- | --- |
| 名称 | 简介 |
| [VerticalScrollPosChanged](#VerticalScrollPosChanged) | 滚动区域垂直滚动位置变化 |
| [HorizontalScrollPosChanged](#HorizontalScrollPosChanged) | 滚动区域水平滚动位置变化 |
| [VisibleItemChanged](#VisibleItemChanged) | 视窗内可见对象范围变化 |

ItemView没有提供单元格被选中, Hover 或者 Click的事件. 因为由使用者提供的Item Factory来创建单元格对象, 用户自己的单元格对象可以自定义各种消息，因为用户自定义item控件的zorder比较高，所以可以优先接收鼠标和键盘消息

成员方法说明

GetDataCount方法

原型

int GetDataCount()

简介

返回当前数据条数. 在表格模式下, 就是行数.

详细说明

这个接口是ItemView为了使用方便加上的. ItemView不会控制数据条目数, 只是简单调用用户设置的DataModel或者DataTable返回数据个数. 使用DataMode的话, 返回DataModel:GetDataCount(); 使用DataTable的话, 返回lua table的行数

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| int | 行数 |

参考

[GetColumnCount](#GetColumnCount)

GetColumnCount方法

原型

int GetColumnCount()

简介

在表格模式下, 返回数据列数

详细说明

这个接口是ItemView为了使用方便加上的. ItemView不会控制数据列数, 只是简单调用用户设置的DataModel或者DataTable返回列数. 使用DataModel的话返回DataModel:GetColumnCount; 使用DataTable的话返回lua table作为二维数组的列数.

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| int | 列数 |

参考

[GetDataCount()](#GetDataCount)

GetViewportSize方法

原型

int, int GetViewportSize()

简介

返回视窗大小.

详细说明

视窗即滚动区域中可见的矩形区域. 由于窗口尺寸限制, ItemView大部分情况下无法同时显示所有数据, 视窗是ItemView可以显示的那部分矩形. 下面图示中红框内部是视窗Viewport, 黑框里面是滚动区域即ScrollArea

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| int | 视窗宽 |
| Int | 视窗高 |

参考

[GetScrollSize](#GetScrollSize) ()

GetFirstVisibleItemIndex方法

原型

int GetFirstVisibleItemIndex()

简介

返回视窗内可见的第一个单元格在数据当中是第几个, 从1开始算

详细说明

见[GetViewportSize](#GetViewportSize)图示, 这个方法返回FirstVisibleItemIndex指向的单元格Index, 即10

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| int | 第一个可见单元格索引 |

参考

[GetViewportSize](#GetViewportSize)

GetScrollSize方法

原型

int,int GetScrollSize()

简介

返回滚动区域尺寸

详细说明

滚动区域即完整显示表格所需要的尺寸. [GetViewportSize](#GetViewportSize)图示中的黑框区域.

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| int | 滚动区域宽 |
| int | 滚动区域高 |

参考

[GetViewportSize](#GetViewportSize)

SetDataTable 方法

原型

void SetDataTable(table dataTable)

简介

设置要显示的数据table

详细说明

以table直接存储要显示的数据, 并把这个table设置给ItemView.

如果调用过SetDataTable, 那么这个调用会覆盖之前设置的table.

如果之前调用过SetDataModel用dataModel来控制显示数据, 那么这个调用会改用table来直接存储数据, 并把ItemView内部之前设置的dataModel置为nil.

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |
| dataTable | table | 无 | 传入 | dataTable[row][column]将会显示在滚动区域的第row行第column列 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |

参考

[SetDataModel](#SetDataModel)

SetDataModel方法

原型

SetDataModel(userdata, callbackTable)

简介

设置用来决定显示数据的dataModel. 由一个方法列表callbackTable和任意类型的用户数据userdata组成

详细说明

通过设置回调方法列表来设置每一行每一列的数据, ItemView显示数据需要的信息都通过callbackTable包含的方法来确定.

如果之前设置过其他的dataModel, 此处dataModel即指callbackTable加上userdata, 那么之前的设置会被覆盖.

如果之前调用过SetDataTable通过table来存储数据, 那么这次调用会改用回调方法来得到数据.

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |
| callbackTable | table | 无 | 传入 | 描述很长, 见下文 |
| userdata | 任意类型 | 无 | 传入 | ItemView回调callbackTable中的方法时传回给使用者, 使用者可以用userdata来存储任何信息. 下文有个例子 |

参数 callbackTable

这是一个table, key是方法名, value是函数, 总览如下:

|  |  |  |  |
| --- | --- | --- | --- |
| 键 | 函数原型 | 必需/可选 | 说明 |
| GetCount | int GetCount(userdata) | 必需 | 返回数据条数(即行数) |
| GetColumnCount | int GetColumnCount(userdata) | 必需 | 返回表格列数 |
| GetItemAtIndex | type GetItemAtIndex(userdata, row, column) | 必需 | 返回第row行第column列的数据 |
| GetColumnNameList | table GetColumnNameList(userdata) | 可选 | 返回Header上的列名 |
| PrepareData | void PrepareData(userdata, from, to) | 可选 | 提前加载数据 |
| ReleaseData | void ReleaseData(userdata, from, to) | 可选 | 释放数据 |
| GetDataBatch | table GetDataBatch(userdata, from, to) | 可选 | 返回第from行到第to行的数据 |
| SetDataBatchReadyListener | void SetDataBatchReadyListener(userdata, function listener) | 如果PrepareData非空, 这两个方法需要提供一个, 否则加载的数据不会显示 | 设置一批数据就绪后的回调函数, 使用者有义务调用listener |
| SetSingleDataReadyListener | void SetSingleDataReadyListener(userdata, function listener) | 设置一个单元格的数据就绪后的回调函数, 使用者有义务调用listener |
| SetDataChangeListener | void SetDataChangeListener(userdata, function listener) | 可选 | 设置一个数据发生变化后通知ItemView的回调函数, 使用者有义务调用它使得ItemView更新 |

ItemView会强制使用者在callbackTable中提供以下方法, 如果没有提供会在设置时报错, 其中参数userdata是在SetDataModel时传入的userdata, 使用者可以用来存放任何想要在回调时参考的数据.

callbackTable.GetCount(userdata), 返回数据条数

callbackTable.GetColumnCount(userdata), 返回数据表格列数

callbackTable.GetItemAtIndex(userdata, row, column), 返回第row行, 第column列要显示的数据

ItemView还会在检查callbackTable是否有以下方法, 有的话就会调用

callbackTable.GetColumnNameList(userdata), 返回header上显示的列名列表, 如果没有这个方法会显示”Section 1, 2, 3...”

如果ItemView的属性PreloadDataCount不为0, ItemView还会检查callbackTable是否含有以下方法:

callbackTable.PrepareData(userdata, from, to), 提前加载从from行(含)到to行(含)的数据, 如果使用者没有提供这个方法, 会循环调用GetItemAtIndex(userdata, row, column)得到from行到to行的所有数据. 在ItemView滚动条位置改变的时候, 如果需要显示的那一行数据还没有被提前加载过, 就会调用PrepareData加载以目标行为中点的PreloadDataCount条数据, 在这PreloadDataCount条数据中之前已经被加载过并且还没有被释放的不会重复加载.

callbackTable.ReleaseData(userdata, from, to), 释放从from行(含)到to(含)的数据, 如果没有这个方法, 就什么都不做. 在ItemView滚动条位置改变并且提前加载数据的时候, 对应的会有另一部分数据被释放, 不管使用者有没有提供ReleaseData, ItemView都会将这部分数据置为nil, 所以下次再需要显示它们的时候还会重新加载.

如果ItemView的属性PreloadDataCount不为0, 并且callbackTable提供了PreloadData, 那么使用者就有义务在数据加载好的时候通知ItemView, 这时候不会强制检查callbackTable是否提供以下方法, 但是如果没有提供的话, 就什么都显示不出来了(因为dataModel承诺准备数据, 但是准备好了没有通知)

callbackTable.SetDataBatchReadyListener(userdata, function dataBatchReadyListener), dataBatchReadyListener是一个方法, 函数原型是function OnDataBatchReady(from, to), ItemView会调用callbackTable.SetDataBatchReadyListener把这个监听者方法传给使用者, 使用者在数据准备就绪之后要调用这个监听者方法通知ItemView从第from行(含)到第to行(含)的数据已经可以显示了.

callbackTable.GetDataBatch(userdata, from, to), ItemView调用callbackTable.PrepareData()提前加载数据, 使用者准备好数据调用上一段介绍的dataBatchReadyListener通知ItemView之后, ItemView会检查callbackTable有没有提供GetDataBatch方法, 如果提供了, 就调用这个方法得到从第from行(含)到第to行(含)的数据, 如果没有提供, 就循环调用GetItemAtIndex(userdata, row, column)得到这些数据.

callbackTable.SetSingleDataReadyListener(userdata, function singleDataReadyListener), ItemView优先调用SetDataBatchReadyListener(), 如果SetDataBatchReadyListener为空, ItemView会调用这个方法传给使用者一个singleDataReadyListner监听者方法. 这个方法的原型是function OnSingleDataReady(row, column), 使用者在每一个单元格的数据准备就绪后, 都要调用一次singleDataReadyListener来通知ItemView第row行第column列的数据已经可以显示了, ItemView会调用GetItemAtIndex(userdata, row, column)来显示这个单元格的数据.

callbackTable. SetDataChangeListener(userdata, function dataChangedListener), 如果dataModel提供这个方法, ItemView会调用它传给使用这个一个在数据发生变化时回调的监听者方法. 这个方法的原型是function OnDataChanged(int from, int to), 表示DataModel中从第from行(含)到第to行(含)需要被更新. 如果使用者设置的PreloadDataCount为零, ItemView会直接调用GetItemAtIndex(row, column)来更新正在被显示的数据, 如果PreloadDataCount>0, ItemView会把已经加载的数据和被更新的数据的交集重新加载(例如已经加载了1~100, dataChanged(80~120), ItemView会重新加载80~100).

参数userdata

ItemView本身对userdata不做任何假设与修改, userdata在ItemView调用callbackTable中的方法时传回给使用者, 便于使用者查询存储信息.

举个例子:

使用者创建了两个ItemView, 它们的数据模块功能相似但不完全相同, callbackTable中包含的方法都可以重用, 但是需要对两个ItemView区别对待, 在SetDataModel的时候, 我们可以把不同的userdata, 相同的callbackTable设置给两个ItemView, 在实现回调方法GetItemAtIndex等函数的时候, 可以通过ItemView传回来的userdata来判断是哪一个ItemView的数据.

Function SetupViewAndModel()

Local itemViewA, itemViewB = ... -- 得到ItemView对象

Local dataModelCallbackTable = {

“GetItemAtIndex” = MyGetItemAtIndex,

“GetCount” = MyGetCount,

“GetColumnCount” = MyGetColumnCount

}

-- 把同一个callbackTable设置给两个View, userdata分别是两个view的名字

itemViewA.SetDataModel(“viewA”, dataModelCallbackTable)

itemViewB.SetDataModel(“viewB”, dataModelCallbackTable)

End -- end of function SetupViewAndModel

ItemView回调GetItemAtIndex的时候, 根据userdata判断是哪个ItemView

function MyGetItemAtIndex(userdata, row, column)

If userdata == “viewA” then return “ViewA”..row..column

If userdata == “viewB” then return “ViewB”..row..column

end -- end of function MyGetItemAtIndex

返回值 (方法SetDataModel的返回值)

|  |  |
| --- | --- |
| 类型 | 描述 |
| 无 |  |

参考[SetDataTable](#SetDataTable)

SetItemFactory方法

原型

SetItemFactory(userdata, name, callbackTable)

简介

设置ItemView要得到单元格对象和单元格显示信息的回调方法.

详细说明

ItemView创建滚动区域内的单元格时, 会调用使用者通过SetItemFactory()设置的回调方法列表中包含的回调方法来创建对象, 设置对象大小位置.

如果callbackTable没有提供创建对象的方法, 那么ItemView会尝试把name当成template name来创建一个实例(调用createInstance), 如果创建失败, 会再尝试把name当成control class name来创建一个对象(调用createUIObject), 如果还是失败了, 那么使用者用错了.

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |
| userdata | 任意类型 | 无 | 传入 | 在ItemView回调callbackTable中的方法时会把这个参数传回给使用者使用, 使用者可以用它来存储任何在被回调时想要使用的信息 |
| name | string | 无 | 传入 | 如果callbackTable没有提供创建对象方法, 会用name来尝试创建 |
| callbackTable | table | 无 | 传入 | 见下文描述 |

参数callbackTable

这是一个table, key是方法名, value是函数, 总览如下:

|  |  |  |  |
| --- | --- | --- | --- |
| 键 | 函数原型 | 必需/可选 | 说明 |
| CreateObject | UIObject CreateObject(userdata, int column) | 可选 | 返回一个第column列的UI对象. 使用者可以根据column不同返回不同对象. 但是ItemView假设每一行同一列的对象类型是相同的, 所以没有row参数 |
| GetRowHeight | int GetRowHeight(userdata) | 可选  没有的话会用默认值50 | 返回行高 |
| GetColumnWidth | int GetColumnWidth(userdata, int column, int widthInAll) | 可选  没有的话会用总宽除以列数 | 返回第column列在总宽为widthInAll的情况下想要有多宽 |
| SetItemPos2 | void SetItemPos(userdata, UIObject itemObj, int left, int top, int width, int height) | 可选  没有的话会直接调用itemObj:SetObjPos(left, top, width, height) | 设置单元格对象位置. 此处给使用者一个最终决定单元格对象位置的机会, 留点padding什么的. |
| SetItemData | void SetItemData(userdata, UIObject, data, row, column) | 必需  没有的话ItemView实在不知道怎么显示数据 | 设置单元格对象要显示的数据. 单元格对象可能是ImageObject, 也可能是TextObject, 更可能是使用者自定义的control object, ItemView调用这个方法, 由使用者来决定传入的数据应该怎样显示.  此处参数data的类型未定, 使用者要自己注意保持dataModel/dataTable返回的数据类型与SetItemData接受的数据类型一致. ItemView只是传递数据, 不保证类型安全. |

返回值(方法SetItemFactory的返回值)

|  |  |
| --- | --- |
| 类型 | 描述 |
| 无 |  |

参考

[SetDataModel](#SetDataModel)的回调方法列表和userdata的使用方法

PageUp方法

原型

void PageUp()

简介

按照当前页面高度向上翻一页

详细说明

页面高度即布局图示中显示表格的高度, 滚动条的Scroll Pos会减去这个高度, 同时滚动区域看起来向下移动这个高度.

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |

参考[PageDown](#PageDown)

PageDown方法

原型

void PageDown()

简介

按照当前页面高度向下翻一页

详细说明

页面高度即布局图示中显示表格的高度, 滚动条的Scroll Pos会加上这个高度, 同时滚动区域看起来向上移动这个高度.

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |

参考[PageUp](#PageUp)

SetScrollBarV方法

原型

void SetScrollBarV(UIObject scrollbar)

简介

设置垂直滚动条对象

详细说明

使用者有两种方式来使用自定义的滚动条对象. 一种是在属性ScrollBarClassNameV中指定用来实例化滚动条对象的control class name. 当使用者想要以其他方式来实例化滚动条对象时, 可以再ItemView初始化时调用这个接口来设置滚动条对象.

无论是通过ItemView的属性值, 还是调用这个接口来设置滚动条对象, ItemView都会对滚动条对象包含的函数做检查, 如果不包含以下函数, 会在设置时报错.

这里对scrollbar包含哪些方法的检查和对[SetItemFactory](#SetItemFactory), [SetDataModel](#SetDataModel)中的参数callbackTable的检查是一样的. 使用者可以定义control class, 并定义名称为下面列表中键值的方法.

|  |  |  |  |
| --- | --- | --- | --- |
| 键 | 函数原型 | 必需/可选 | 说明 |
| GetScrollPos | int GetScrollPos() | 必需 | 返回当前滚动条的位置 |
| SetScrollPos | SetScrollPos(int pos) | 必需 | 设置当前滚动条位置为pos |
| GetScrollRange | int min, int max GetScrollRange() | 必需 | 返回滚动条最大最小范围 |
| SetScrollRange | SetScrollRange(int min, int max) | 必需 | 设置滚动条范围 |

例子

<control class=”MyScrollBar”>

<!-- 别的 -->

<method\_def>

<SetScrollPos file="XXX" func="XXX"/>

<GetScrollPos file="XXX" func="XXX"/>

<SetScrollRange file="XXX" func="XXX"/>

<GetScrollRange file="XXX" func="XXX"/>

</method\_def>

</control>

local scrollBar = objFactory:CreateUIObject(nil, “MyScrollBar”)

itemView:SetScrollBarV(scrollbar)

函数参数 (SetScrollBarV的函数参数)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |
| Scrollbar | UIObject | ItemView.ScrollBarV | 传入 | 垂直方向滚动条对象. 布局图示中Vertical Scroll Bar区域 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |

参考[SetScrollBarH](#SetScrollBarH)

SetScrollBarH方法

原型

void SetScrollBarH(UIObject scrollBar)

简介

设置水平滚动条对象

详细说明

见[SetScrollBarV](#SetScrollBarV)的详细说明. 除了方向不一样, 二者的使用方法是一样的.

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |
| Scrollbar | UIObject | ItemView.ScrollBarH | 传入 | 水平方向滚动条对象. 布局图示中Horizontal Scroll Bar区域 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |

参考[SetScrollBarV](#SetScrollBarV)

SetHeader方法

原型

void SetHeader(UIObject header)

简介

设置Header对象.

详细说明

使用者有两种方式来使用自定义的Header对象. 一种是在属性HeaderClassName中指定用来实例化滚动条对象的control class name. 当使用者想要以其他方式来实例化滚动条对象时, 可以再ItemView初始化时调用这个接口来设置Header对象.

无论是通过ItemView的属性值, 还是调用这个接口来设置滚动条对象, ItemView都会对滚动条对象包含的函数做检查, 如果不包含以下函数, 会在设置时报错.

这里对header包含哪些方法的检查和对[SetScrollBarV](#SetScrollBarV)中的参数scrollbar的检查是一样的. 使用者可以定义control class, 并定义名称为下面列表中键值的方法.

|  |  |  |  |
| --- | --- | --- | --- |
| 键 | 函数原型 | 必需/可选 | 说明 |
| GetScrollPos | int GetScrollPos() | 必需 | 当水平滚动条位置变化时, Header会一起移动以保证列名和列对齐.  返回当前水平偏移的位置 |
| SetScrollPos | SetScrollPos(int pos) | 必需 | 设置Header的水平偏移为Pos |
| GetSectionNameList | table GetSectionNameList() | 必需 | 返回Header上显示的列名列表 |
| SetSectionNameList | SetSectionNameList(table namelist) | 必需 | 设置Header上显示的列名列表 |

函数参数 (方法SetHeader的函数参数)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |
| Header | UIObject | ItemView.Header | 传入 | 布局图示中Header的区域 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |

参考[SetSectionNameList](#SetHeaderNameList)

SetHeaderNameList方法

原型

void SetHeaderNameList(table namelist)

简介

设置Header区域每一列的列名. Namelist[column]表示第column列的列名.

详细说明

在[SetDataModel](#SetDataModel)(userdata, callbackTable)时, 我们会检查callbackTable中有无GetColumnNameList方法, 如果有的话会调用callbackTable.GetColumnNameList来确定header上显示的列名. 此处SetHeaderNameList(namelist)方法也会确定header上显示的列名. 二者的作用是一样的, 后调用的覆盖先调用的. 重复功能的接口原因如下: Header上显示的列名作为数据结构的一部分, 本应当由DataModel提供, 考虑到使用者还要为此添加一个函数实现, 提供一个方便点的接口直接设置.

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |
| namelist | table | Nil | 传入 | Namelist[i]表示表头上第i个section上显示的文字 |

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |

参考

[SetHeader](#SetHeader), [SetDataModel](#SetDataModel)

事件说明

VerticalScrollPosChanged事件

原型

void VerticalScrollPosChanged(int oldPos, int newPos)

简介

滚动区域垂直方向滚动值发生变化

详细说明

如果用户不想自己设置ScrollBar, 但是又需要监听ItemView的滚动事件, 可以监听这个事件

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |
| oldPos | Int | 无 | 传入 | 之前的滚动位置 |
| newPos | Int | 无 | 传入 | 新的滚动位置 |

返回值

参考 [HorizontalScrollPosChanged](#HorizontalScrollPosChanged)

HorizontalScrollPosChanged事件

原型

void HorizontalScrollPosChanged(int oldPos, int newPos)

简介

滚动区域水平方向位置发生变化

详细说明

如果用户不想自己设置ScrollBar, 但是又需要监听ItemView的滚动事件, 可以监听这个事件

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |
| oldPos | Int | 无 | 传入 | 之前的滚动位置 |
| newPos | Int | 无 | 传入 | 新的滚动位置 |

返回值

参考 [VerticalScrollPosChanged](#VerticalScrollPosChanged)

VisibleItemChanged事件

原型

void VisibleItemChanged(int newFirst, int newLast)

简介

在视窗内可见的Item索引发生变化时这个Event被触发

详细说明

参考下面的图示. 滚动区域滚动到这个位置时, 会触发VisibleItemChanged事件, 第一个可见Item是10, 最后一个是22. 即在视窗内只显示一部分的Item也算可见, 因为需要被渲染.

这个事件被触发的频率在滚动时是很高的. 当视窗上边缘消失/出现一个Item, 或者下边缘出现/消失一个Item时, 这个事件都会被触发. 所以可能有滚动一行高, 但是事件被触发两次的情况.

函数参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 传入传出 | 描述 |
| newFirst | Int | 无 | 传入 | 第一个Item |
| newLast | Int | 无 | 传入 | 最后一个item |

返回值

参考

ItemView Control Class使用用例

文件说明

使用者Xar包文件列表, 不包括ItemView定义文件:

XAR/MyBolt/layout/MainWnd.xml 主界面模板定义

XAR/MyBolt/layout/MainWnd.xml.lua 主界面模板方法定义

XAR/MyBolt/luacode/TestDataModel DataModel用例

XAR/MyBolt/luacode/ItemFactory ItemFactory用例

以下文件定义了ItemView默认使用的ScrollBar和Header, 使用者可以替换成自己定义的控件定义, 在使用ItemView的时候把属性名称换成自己的就可以了.

XAR/MyBolt/layout/ItemViewHeader ItemView.Header控件定义

XAR/MyBolt/layout/ItemViewScrollBar ItemView.ScrollBarV, ItemView.ScrollBarH控件定义

以下文件是ItemView不支持外部替换的子控件定义, 当然使用者可以修改ItemView控件本身的实现来使用自定义的子控件

XAR/MyBolt/layout/ScrollArea.xml 滚动区域控件

XAR/MyBolt/layout/ScrollArea.xml.lua 滚动区域显示相关方法定义

XAR/MyBolt/layout/DataModel.lua 滚动区域数据相关方法定义

XAR/MyBolt/layout/ItemContainer.xml 每一行的单元格对象放在一个容器中, 这是容器

MainWnd.Xml说明

MainWnd.xml的主要控件就是一个ItemView. 在使用时可以指定BackgroundTexture, HeaderHeight, ScrollBarSize等显示属性, ScrollBarClassNameV/H, HeaderClassName等控件类, 还可以指定提前加载的数据条目数PreloadDataCount. 这些在ItemView的定义中都由默认值所以不是必需指定的.

在ItemView的事件OnInitControl中我们设置ItemView使用的DataModel和ItemFactory. 这是建议设置Item Factory和DataModel的时机. OnInitControl是引擎元对象的事件.

除了OnInitControl, 我们在例子中还在eventlist中添加了ItemView自定义事件[VerticalScrollPosChanged](#VerticalScrollPosChanged), Control自定义的事件可以像元对象事件一样在此被监听, 注意事件参数的获取方式和顺序. 在OnInitControl\_ItemView的方法实现中, [HorizontalScrollPosChanged](#HorizontalScrollPosChanged)事件以AttachListener的方式被监听, 这样做的好处是可以动态地去掉对事件的监听.

<xmlcode>

<obj id="itemView" class="ItemView">

<attr>

<left>10</left>

<top>50 + 30</top>

<width>father.width-20</width>

<height>father.height-130</height>

<!-- 以下属性如果用户不设置,会用默认值 -->

<BackgroundTexture></BackgroundTexture>

<ScrollBarClassNameV>ItemView.ScrollBarV</ScrollBarClassNameV>

<ScrollBarClassNameH>ItemView.ScrollBarH</ScrollBarClassNameH>

<ScrollBarSize>20</ScrollBarSize>

<HeaderClassName>ItemView.Header</HeaderClassName>

<HeaderHeight>30</HeaderHeight>

<!-- 提前加载数据条数 -->

<PreloadDataCount>50</PreloadDataCount>

</attr>

<eventlist>

<event name="OnInitControl" file="MainWnd.xml.lua" func="OnInitControl\_ItemView" />

<event name="VerticalScrollPosChanged">

-- 监听自定义控件自定义事件方法一:

- 请注意参数获取及参数顺序

local arg={...}

local itemViewObj = arg[1]

local eventName = arg[2]

local oldPos = arg[3]

local newPos =arg[4]

</event>

</eventlist>

</obj>

</xmlcode>

DataModel说明

XAR/MyBolt/luacode/TestDataModel是一个使用DataModel即callbackTable+userdata来控制显示数据的例子. 在方法GetTestDataModelObject(name)中我们返回一个包含一些方法(见[SetDataModel](#SetDataModel)方法说明)的callbackTable和一个含有c++ class的lua封装类的userdata. callbackTable的方法定义既可以是局部匿名方法也可以指向其他方法, 只要在table中的键值正确就行. Userdata中的lua封装类用来调用在dataModel项目中注册到lua环境中的LuaDataModelClass<XmlDataModel>.

<luacode>

-- callbackTable.SetDataBatchReadyListener =

-- function (userdata, dataBatchReadyListener)

-- userdata.class:AttachDataBatchReadyListener(userdata.className , dataBatchReadyListener)

-- end

callbackTable.SetSingleDataReadyListener =

function(userdata, singleDataReadyListener)

userdata.class:AttachSingleDataReadyListener(userdata.className , singleDataReadyListener)

end

</luacode>

在callbackTable中我们指定了SetSingleDataReadyListener而注释掉了指定SetDataBatchReadyListener的段落. 在这种情况下ItemView会调用SetSingleDataReadyListener传给使用者一个回调方法. 使用者在单条数据(某行某列)就绪之后有义务调用该回调方法通知ItemView, ItemView会显示这条数据(如果它在被显示区域的话). 我们看到这个回调方法作为参数直接传给了c++的lua封装类, 在类模板LuaDataModelClass中可以追踪到在c++中回调lua方法的一种方式.

如果我们去掉指定SetDataBatchReadyListener的段落, ItemView会优先调用它. 在这种情况下使用者得到的回调方法会通知ItemView从某行到某行的一批数据就绪, lua封装类LuaDataModelClass<XmlDataModel>也处理了这种情况. 可以把注释去掉跟踪一下.

XmlDataModel是一个和lua和ItemView都没有关系的解析xml的c++类. LuaDataModelClass类模板处理c++和lua的交互, callbackTable的一众函数才是针对ItemView的需要而定义的. 对于简单的业务逻辑, 调用c++的类不是必需的, 类模板更没有必要. 模板为演示ItemView的使用方法而注册了作为DataModel可能会被使用的每一个函数, 在实际应用时建议使用者定义自己的lua封装以使代码更简洁.

ItemFactory说明

XAR/MyBolt/luaCode/ItemFactory是一个使用ItemFactory来控制显示方式的例子. 在方法GetItemFactory中我们返回一个包含一些方法(见[SetItemFactory](#SetItemFactory)方法的详细说明)的callbackTable和一个空userdata. 可以看到只要使用者在回调方法被调用时自己不访问userdata, userdata对ItemView完全没有影响, 这个对象时给使用者自己用的.