[创建一个简单的树组件 1](#_Toc6406)

[Tree Panel内部使用Tree Store存储数据 2](#_Toc7239)

[tree的根节点不是必须在初始化时设定 4](#_Toc26970)

[Loading the Entire Tree 一次加载 9](#_Toc21261)

[Saving Data 保存数据 11](#_Toc25312)

Tree Panel和Grid Panel继承自相同的基类，所以所有从Grid Panel能获得到的特性、扩展、插件等带来的好处在Tree Panel中也同样可以获得。列、列宽调整、拖拽、渲染器、排序、过滤等特性，在两种组件中都是差不多的工作方式

## 创建一个简单的树组件

Ext.create('Ext.tree.Panel', {

renderTo: Ext.getBody(),

title: 'Simple Tree',

width: 150,

height: 150,

root: {

text: 'Root',

expanded: true,

children: [{

text: 'Child 1',

leaf: true},

{

text: 'Child 2',

leaf: true

},

{

text: 'Child 3',

expanded: true,

children: [

{

text:'Grandchild',

leaf: true}]

}]

}

});

Tree Panel内部使用Tree Store存储数据。上面的例子中使用了root配置项作为使用store的捷径。如果我们单独指定store，代码像这样

外观定制

把useArrows设置为true，Tree Panel就会隐藏前导线使用箭头表示节点的展开

columns 多列

由于Tree Panel也是从Grid Panel相同的父类继承的，因此实现多列很容易。

ar tree = Ext.create('Ext.tree.Panel', {

renderTo: Ext.getBody(),

title: 'TreeGrid',width: 300,

height: 150,

fields: ['name', 'description'],

//注意这里

columns: [{

xtype: 'treecolumn',

text: 'Name',

dataIndex: 'name',

width: 150,

sortable: true

}, {

text: 'Description',

dataIndex: 'description',

flex: 1,

sortable: true

}],

root: {

name: 'Root',

description:'Root description',

expanded: true,

children: [{

name: 'Child 1',

description: 'Description 1',

leaf: true

}, {

name: 'Child 2',

description: 'Description 2',

leaf: true

}]

}});

这里面的columns配置项期望得到一个Ext.grid.column.Column配置，就跟GridPanel一样的。唯一的不同就是Tree Panel需要至少一个treecolumn列

如果不配置column，tree会自动生成一列treecolumn，并且它的dataIndex是text，并且也自动隐藏了表头，如果想显示表头，可以用hideHeaders配置为false。

添加节点

tree的根节点不是必须在初始化时设定。后续再添加也可以：

var tree = Ext.create('Ext.tree.Panel');

tree.setRootNode({

text: 'Root',

expanded: true,

children: [{

text: 'Child 1',

leaf: true

}, {

text: 'Child 2',

leaf: true

}]

});

尽管对于很小的树只有默认几个静态节点的，这种直接在代码里面配置的方式很方便，但是大多数情况tree还是有很多节点的。让我们看一下如何通过程序添加节点

var root = tree.getRootNode();

var parent = root.appendChild({

text: 'Parent 1'

});

parent.appendChild({

text: 'Child 3',

leaf: true

});

parent.expand();

有时我们期望将节点插入到一个特定的位置，而不是在最末端添加。除了appendChild方法，Ext.data.NodeInterface还提供了insertBefore和insertChild方法

var child = parent.insertChild(0, {

text: 'Child 2.5',

leaf: true

});

parent.insertBefore({

text: 'Child 2.75',

leaf: true},

child.nextSibling

);

insertChild方法需要一个节点位置，新增的节点将会插入到这个位置。insertBefore方法需要一个节点的引用，新节点将会插入到这个节点之前

NodeInterface Fields

使用tree数据的时候，最重要的就是理解NodeInterface是如何工作的。每个tree节点都是一个用NodeInterface装饰的Model实例。假设有个Person Model，它有两个字段id和name：

Ext.define('Person', {

extend: 'Ext.data.Model',

fields: [

{ name: 'id', type: 'int' },

{ name: 'name', type: 'string' }]

});

果只做这些，Person Model还只是普通的Model，如果取它的字段个数：

console.log(Person.prototype.fields.getCount()); //输出 '2'

但是如果将Person Model应用到TreeStore之中后，就会有些变化：

var store = Ext.create('Ext.data.TreeStore', {

model: 'Person',

root: {name: 'Phil'}

});

console.log(Person.prototype.fields.getCount()); //输出 '24'

默认持久化的：

1. parentId - 用来指定父节点的id，这个字段应该总是持久化，不要覆盖它
2. leaf - 用来指出这个节点是不是叶子节点，因此决定了节点是不是可以有子节点，最好不要改变它的持久化设置

默认不持久化的：

1. index - 用来指出当前节点在父节点的所有子节点中的位置，当有节点插入或者移除，它的所有邻居节点的位置都会更新，如果需要，可以用这个属性去持久化树节点的排列顺序。然而如果服务器端使用另外的排序方法，最好把这个字段保留为非持久化的，当使用WebStorage Proxy作为存储，且需要保留节点顺序，那一定要设置为持久化的。如果使用了本地排序，建议设置非持久化，因为本地排序会改变节点的index属性
2. depth 用来存储节点在树中的层级，如果server需要保存节点层级请开启持久化。使用WebStorage Proxy的时候建议不要持久化，会多占用存储空间。
3. checked 如果在tree使用checkbox特性，看业务需求来开启持久化
4. expanded 存储节点的展开收起状态，要不要持久化看业务需求
5. expandable 内部使用，不要变更持久化配置
6. cls 用来给节点增加css类，看业务需求
7. iconCls 用来给节点icon增加css类，看业务需求
8. icon 用来自定义节点，看业务需求
9. root 对根节点的引用，不要变动配置
10. isLast 标识最后一个节点，此配置一般不需要变动
11. isFirst 标识第一个节点，此配置一般不需要变动
12. allowDrop 用来标识可放的节点，此配置不要动
13. allowDrag 用来标识可拖的节点，此配置不要动
14. loaded 用来标识子节点是否加载完成，此配置不要动
15. loading 用来标识子节点是否正在加载中，此配置不要动
16. href 用来指定节点链接，此配置看业务需求变动
17. hrefTarget 节点链接的target，此配置看业务需求变动
18. qtip 指定tooltip文字，此配置看业务需求变动
19. qtitle指定tooltip的title，此配置看业务需求变动
20. children 内部使用，不要动

Loading Data 加载数据

有两种加载数据的方式。一次性加载全部节点和分步加载，当节点过多时，一次加载会有性能问题，而且不一定每个节点都用到。动态分步加载是指在父节点展开的时候加载子节点。

## Loading the Entire Tree 一次加载

Tree的内部实现是只有节点展开的时候加载数据。然而全部的层级关系可以通过一个嵌套的数据结构一次全部加载，只要配置root节点是展开的即可

Ext.define('Person', {

extend: 'Ext.data.Model',

fields: [

{ name: 'id', type: 'int' },

{ name: 'name', type: 'string' }],

proxy: {

type: 'ajax',api: {

create: 'createPersons',

read: 'readPersons',

update: 'updatePersons',

destroy: 'destroyPersons'

}}

});

var store = Ext.create('Ext.data.TreeStore', {

model: 'Person',

root: {

name: 'People',

expanded: true

}

});

Ext.create('Ext.tree.Panel', {

renderTo: Ext.getBody(),

width: 300,

height: 200,

title: 'People',

store: store,

columns: [{

xtype: 'treecolumn',

header: 'Name',

dataIndex: 'name',

flex: 1

}]

});

## Saving Data 保存数据

创建、更新、删除节点都由Proxy自动无缝的处理了。

Creating a New Node 创建新节点

var newPerson = Ext.create('Person', { name: 'Nige', leaf: true });
store.getNodeById(2).appendChild(newPerson);

由于Model中定义过proxy，Model的save方法可以用来持久化节点数据：

newPerson.save();

Updating an Existing Node 更新节点

Store.getNodeById(1).set('name', 'Philip');

Removing a Node 删除节点

store.getRootNode().lastChild.remove();

Bulk Operations 批处理

也可以等创建、更新、删除了若干个节点之后，由TreeStore的sync方法一次 保存全部

store.sync();