[Обучение Ext JS] :: Часть 8. Ext JS растет на деревьях

Фев 28 2010

Фильтры

В **Ext JS** 2.2, класс Ext.tree.TreeFilter отмечен как "experimental" (экспериментальный), поэтому его мы только затронем. и не будем углубляться. Он создан для тез случаев где пользователю понадобится найти узел ориентируясь по заданному атрибуту. Им может быть текст, ID или любые данные отправленные в него при создании. Давайте возьмем свежесозданное контекстное меню и воспользуемся им в демонстрационных целях. Сначала нам надо создать TreeFilter:

```
var filter = new Ext.tree.TreeFilter(tree);
```

Вам необходимо вернуться к настройкам контекстного меню и добавить новую запись в элементы свойств настроек:

```
{ text: 'Filter', handler: filterHandler }
```

Теперь нам надо создать функцию filterHandler которая выполняет роль фильтра:

```
function filterHandler() {
   var node = tree.getSelectionModel().getSelectedNode();
   filter.filter('Bee', 'text', node);
}
```

Как и с другими нашими функциями обработчиков, мы начинаем с получения выбранного узла дерева, и затем вызываем функцию фильтра.У функции есть три аргумента:

1. Значение подлежащее фильтрации by
2. Атрибут подлежащий фильтрации. Это опционально
3. Узел с которого начинается процедура фильтрации

Мы пометили выбранный узел как начальный. Это значит, что к узла, на котором мы щелкнули правой кнопкой мыши для того чтобы вызвать всплывающее меню, все дети будут отфильтрованы по указанным значениям.

Нашим примерам, разумеется, не требуется такой уровень фильтрации, но есть ситуации, когда это может оказаться довольно полезной особенностью. Онлайн документация программного обеспечения, с несколькими уровнями деталей может быть представлена как дерево, и TreeFilter моет помочь найти нужную тему. В более сложном случае вы можете использовать чекбоксы или всплывающие диалоговые окна для ввода значений фильтра, что сделает всю конструкцию более гибкой.

Корни

Хотя мы и показали несколько мощнейших техник использования поддержки дерева **Ext**, его настоящая сила лежит в здоровье настроек, методов и точек привязки, которые дают многочисленные классы. Мы уже ознакомились с различными способами настроек классов TreePanel и TreeNode, что дает нам доступ к некотором значительным возможностям. Тем не менее есть еще настройки, которые можно использовать для улучшения дерева, и мы собираемся рассмотреть самые интересные из них.

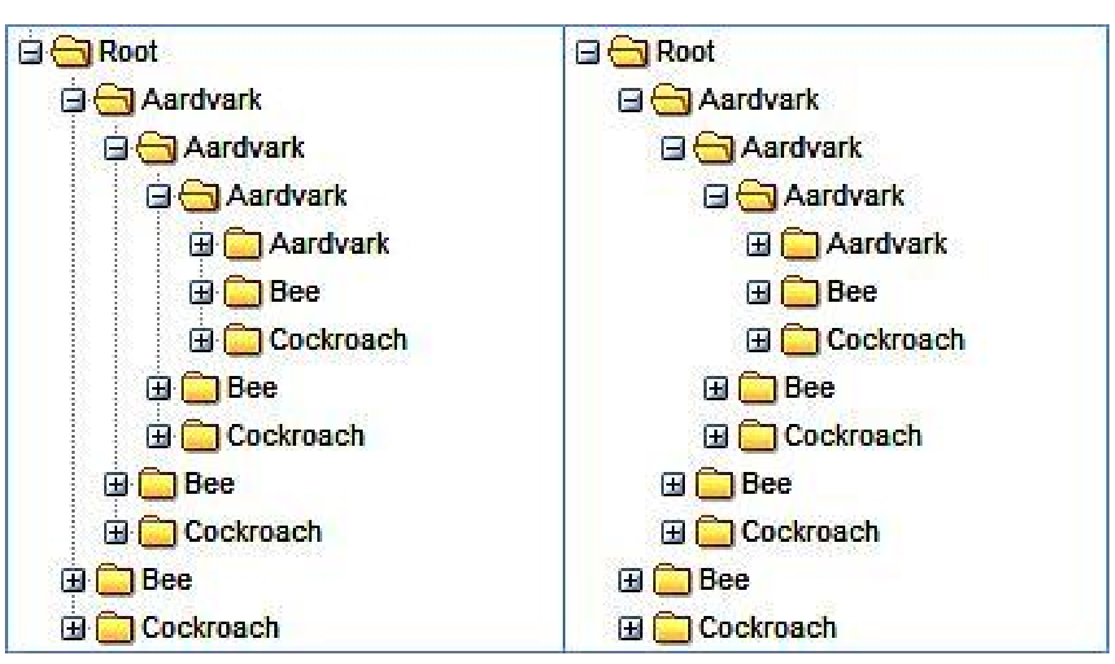
Вылизываем TreePanel

По умолчанию существует несколько графических улучшений для TreePanel, которые, в зависимости от требований вашего приложения, могут быть нежелательны. Например установка анимации на false предотвратит использование сглаженных эффектов анимации при развертывании и свертывании узлов. Это может быть полезно в ситуациях, при которых узлы постоянно сворачиваются и разворачиваются, и анимация, замедляющая работу, может выводить из себя.

Поскольку TreePanel происходит из Ext.Panel, то она поддерживает все стандартные особенности панели. Легко запомнить, поскольку это значит, что поддерживаются панели инструментов сверху и снизу, раздельные заголовки и колонтитулы и свертывание/развертывание элементов панели. также можно включить TreePanel в любую Ext.ViewPort или Ext.layout.

Косметика

Говоря о косметике, TreePanel позволяет подретушировать пару линий, которые, если выставить им значение false, отключат линии, показывающие отношения в иерархии дерева. Это может быть полезно в тех случаях, когда вы создаете очень простое дерево в котором линии засоряют интерфейс.



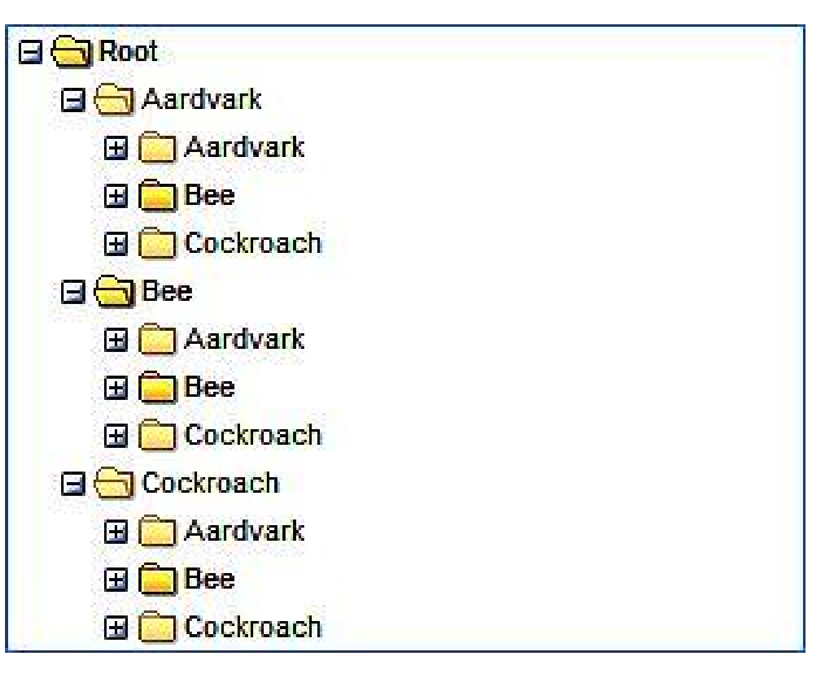
hlColor применимо для деревьев с включенным drag-and-drop, и отвечает за начальную подсветку выделения, которое включается, когда узел отпускается. Эту опцию можно полностью отключить, выставив ddDrop на false. А установив Setting trackMouseOver на false, вы отключите выделение появляющиеся при наведении курсора на узел.

Точная настройка TreeNode

Во многих случаях вы не будете собственноручно создавать TreeNodes, а только ваш корневой узел. Таким образом вы можете подумать, что опции настроек для вас не важны (и даже бесполезны). Не совсем. Учитывая что вы используете для генерации узлов не просто свойства текста и id получаемые от **JSON** — Любое свойство, подходящее по параметрам, в **JSON** будет использовано для создания узлов. Если у вас примерно такой **JSON**:

```
[
  { text: 'My Node', disabled: true, href: 'http://extjs.com'}
]
```

У вас будет узел, который изначально отключен, но при включении он будет действовать как ссылка на сайт **Ext JS**.



Эта особенность чрезвычайно полезна для отправления информации по приложениям на ваши TreeNodes. Например, ваша серверная логика может говорить что определенные узлы не могут иметь детей. Установив allowChildrenfalse значит, что узел не может быть использован как цель для сброса, для перетаскиваемого узла. Таким же образом, вы можете запретить отдельным узлам быть перетаскиваемыми используя draggable: false. мы можем выставить статус чекбокса на узле при помощи checked: true. На самом деле мы просто уточняем, является эта опция true или false. Но это приведет к тому, что рядом с узлом появится чекбокс. Эти опции настроек позволяют вам определить поведение ваших узлов, основываясь на некоторой серверной логике, но не требуют ручной обработки. Существует еще несколько полезных опций настроек, доступных для TreeNode. Вы можете присвоить свои иконки, использую опцию иконок, или сделать стайлинг **CSS** используя опцию cls. Опция qtip позволяет создать всплывающую подсказку, возможно с описанием содержимого узла.

Управление

Как только TreePanel была настроена, мы можем начать работу с узлами. Панель позволяет переходы по иерархии, начиная с выбранного узла и двигаясь от родителей (или детей) вверх или вниз по ветви. Мы можем также выбирать узлы и разворачивать их путь, что может быть полезно при поиске определенного узла.

Методы expandAll и collapseAll довольно красноречивы и говорят сами за себя. каждый метод использует логический параметр для того, чтобы определиь - нужна анимация или нет.

первый параметр метода expandPath, это собственно "путь" к узлу. Путь является уникальным идентификатором узла в иерархии и берется из строки, которая полностью описывает местоположение узла в дереве. Например вот так:

```
/n-15/n-56/n-101
```

Здесь у нас представление местоположения узла с ID n-101. n-15 является корневым узлом, с ребенком n-56; а n-101 в свою очередь, ребенок n-56. Если вы знакомы с XPath, то эта система вам хорошо известна. Если же нет, то читайте это почтовым или IP адресом. Уникальной отметкой узла.

Отправляем это значение к expandPath, дерево доберется до указанного узла, открывая необходимые ветви. Представьте следующий код:

Исходный код примера:

Исходный код примера:

```
Ext.Msg.prompt('Node', 'Please enter a product name',
            function(btn, text){
   if (btn == 'ok'){
      var path = GetNodePathFromName(text);
    tree.expandPath(path);
   }
});
```

Функция GetNodePathFromNameможет выполнить поиск ID узла по серверу, позволяя быстрые переходы по структуре дерева. Или, с той же целью можно использовать TreePanel.getNodeById. Вместо того чтобы разворачивать узел с ним можно работать дальше.

В некоторых обстоятельствах вам может понадобиться совершить обратное действие, когда у вас есть узел, но нужен его путь. для этой цели прекрасно служит TreeNode.getPath. Его можно использовать как способ сортировки местоположения узла.

Дальнейшие методы

У TreeNode есть несколько других полезных методов. Мы уже рассмотрели sort и remove, но сейчас мы можем добавить некоторые основные полезные методы, например collapse и expand, enable и disable, а также полезные дополнения: expandChildNodes и collapseChildNodes, которые пересекают все дочерние узлы третейского корня и изменяют их состояния расширений. Методы findChild и findChildBy позволяют простой и настраиваемый поиск дочерних узлов, как показано в следующем примере мы ищем первый узел с параметром цены 300:

```
var node = root.findChild('price', 300);
```

В некоторых случаях вам придется управлять большим количеством параметров узлов в иерархии. Это можно сделать используя метод TreeNode.eachChild:

```
root.eachChild(function(currentNode) {
   currentNode.attributes.price += 30;
});
```

поскольку первый параметр в eachChild является функцией, мы можем выполнить любую логику, требуемую для нашего приложения.

Захват событий

мы уже продемонстрировали пару методов отслеживания взаимодействия пользователя с деревом, но есть еще много событий, которые можно использовать как крюки для вашего кода. Ранее мы обсуждали использование клетчатые опции настроек в TreeNode. Когда включен чекбокс узла, запускается событие checkchange. Это может быть очень полезно для визуального выделения:

```
tree.on('checkchange', function(node, checked) {
  node.eachChild(function(currentNode) {
     currentNode.ui.toggleCheck();
  });
}
```

мы передаем отметку через детей TreeNode. мы также можем выделить эти узлы вопросом, для более ясного отображения того, что их статус изменился. Или сделать что-нибудь еще, например добавить информацию о свежеотмеченных узлах и вывести эту информацию где-нибудь на странице.

Наиболее распространенное использование событий TreePanel заключается в подтверждении изменений или их сохранение в серверном хранилище. например дерево категоризированных продуктов может иметь некоторые логические ограничения, определенные категории могут указывать максимальную цену на продукт. Мы можем использовать событие beforeappend для проверки этого:

```
tree.on('beforeappend', function(tree, parent, node) {
   return node.attributes.price < parent.attributes.maximumPrice;
});
```

Этот пример показывает модель, которую вы видели на протяжении всего **Ext JS** — возврат false от обработчика событий отменяет действие. В этом случае, если цена добавленного узла больше цены родителя, то узел не будет добавлен.

Состояние запоминания

Во многих приложениях TreePanels используется для упрощения навигации, показывая иерархическую структуру с узлами, которые являются ссылками на страницу содержащую более детальную информацию по узлу. В этом случае если пользователь хочет просмотреть несколько страниц, одну за другой, прекрасное поведение TreePanel может вызвать депрессию. А все потому, что дерево не сохраняет состояния между обновлениям страницы, таким образом любой развернутый узел будет считаться свернутым, если пользователь захочет вернуться. Если же пользователю надо углубиться по ветке и каждый раз.ю возвращаясь назад они будут сразу к дереву, а не просмотренному узлу... не думаю, что их терпения хватит надолго.

StateManager

Теперь, когда мы уже умеем управляться с TreePanel, выяснение того как сохранять и восстанавливать состояния покажется нам очень простым занятием. Самое важное, что нам предстоит сделать, это записать каждое развертывание TreeNode, и затем, когда страница перезагружается, "воспроизвести" эти развертывания. Для сохранения мы можем использовать Ext.state.Manager с CookieProvider. Запускается это вот так:

```
Ext.state.Manager.setProvider(new Ext.state.CookieProvider());
```

Теперь нам надо установить что именно мы будем хранить и логично,если это будет путь последнего раскрытого узла. Это значит, что можем просто развернуть путь и показать пользователю последний просмотренный элемент иерархии. Вот несколько наивный способ осуществления этой идеи:

```
tree.on('expandnode', function (node){ Ext.state.Manager.
set(“treestate", node.getPath());
});
```

В этомкоде мы просто обрабатываем событие expandnode и записываем путь любого раскрытого узла, используя TreeNode.getPath. Поскольку мы переписываем значения при каждом просмотре, treestate должен содержать путь до последнего просмотренного узла. Затем мы можем проверить это значение при загрузке страницы:

```
var treeState = Ext.state.Manager.get(“treestate");
if (treeState)
  tree.expandPath(treeState);
```

Если treestate предварительно было записано, мы можем использовать все это, для развертывания дерева до последнего просмотренного узла.

Предостережения

Как я уже сказал, этот код наивный, то есть, он не обрабатывает случаи когда пользователь просматривает, затем сворачивает узел, а потом уходит куда-то далеко, и возвращается. В таких случаях свертывание узла не будет сохранено. Поэтому, когда мы восстановим данные, пользователь увиит узел развернутым. Обрабатывая событие collapsenode мы можем справиться с этой проблемой. Так же у нас есть трудности с одновременным просмотром нескольких узлов. Если открыта больше чем одна ветвь, наш код запомнит лишь самое последнее действие. хранения массива просмотренных узлов мы рассмотрим позже.

Заключение

В этой главе мы показали сильные стороны TreePanel, и хотя не все из них легко использовать, они все же пригодятся для создания функционального приложения.

Использование асинхронной загрузки - важная особенность TreePanel, поскольку дает возможность перерабатывать огромные количества динамических данных. Она так же явно обрабатывается Ext.tree, а значит его применение будет полезно не только пользователям, но и разработчикам

Не смотря на всю свою силу, классы Ext.tree до сих пор кажутся легковесными. И эту силу легко приручить, используя опции настроек, методы и события которые предоставляют TreePanel и TreeNode. TreeSorter и TreeNodeUI ключевые элементы головоломки, добавляя функционал и позволяя настроить вид.

Поскольку Ext.TreePanel расширяет Panel, которая в свою очередь расширяет BoxComponent, мы получаем все эти сильные элементы и поддержку схемы компоновки которые есть в **Ext JS**. Поддержка BoxComponent будет особенно интересна когда мы двинемся дальше, поскольку это означает, что деревья могут быть с легкостью включены в различные настройки в пределах Ext.Window. Что и будет нашей следующей темой.

- « первая
- ‹ предыдущая
- 1
- 2
- 3

- English

Голос
24