



ExtJS 4.1会带来什么

Posted 周五, 01/13/2012 - 00:04 by [admin](#)

即将到来的ExtJS 4.1的焦点是性能。我们一直在为这努力工作，而这主要集中在两个方面：渲染和布局。虽然我们的大多数时间一直致力于这项努力，但也有

很多其他方法的进展可以分享。这些改进当中，主要的改进包括Grid、BorderLayout和海王星主题预览这些内容。

性能

提高性能的前提是必需去测量它。因此，要成功地和永久地提高性能，测量已成为定期构建和测试过程的一部分。这是我们要为ExtJS 4.1做的第一件事。如为了在常用配置内实现动态跟踪，我们创建了连续的基础上使用的简单测试工具。我们使用这些工具来跟踪每次构建的关键指标。

我们跟踪对应页面的生命周期性能指标：加载、初始化、渲染和布局。

加载

一个ExtJS应用的生命周期是从调用“onReady”函数开始。在这之前，有很多事情发生。当我们说“页面加载”时，可能意味着很多不同的事情，但为简单起见，这里我们定义为在执行“ext-all.js”的第一行代码开始，任何onReady函数被调用之前为终点这段时间内。这里主要包括执行Ext.define语句填充Ext的命名空间，及检测页面DOM已经准备好的时间。

初始化

当onReady函数被调用时，应用程序开始接管。应用程序根据需要开始处理自定义初始化，而其主要的工作是创建要显示的组件和容器。在某些应用，会创建上百个组件。而这些组件或容器之间有些会一起创建、初始化和连接。

在ExtJS 4，组件很多时候需要与之前版本作比较。譬如，面板的标题栏构成。Header组件实际上是一个容器，它包含一个基本标题组件和一套工具组件（可选），而这些组件是使用HBoxLayout进行管理的。这意味着可以很容易的面板标题栏添加组件，也意味着相同的面板配置，在ExtJS 4中会创建更多的组件和容器。纵观ExtJS 3的主题示例，有在50个容器内有148个组件。而相同的配置，在ExtJS 4，会在97个容器内产生271个组件。这使得这方面的优化至关重要。

（注：ExtJS 4性能比ExtJS 3慢的一个主要原因就是这个问题，ExtJS 4中，很多小容器的小部件都组件化了，这增加了其灵活性和易用性，但性能会有损失。最典型的例子其实是面板DockLayout的使用，有兴趣可以在这方面研究一下）

渲染

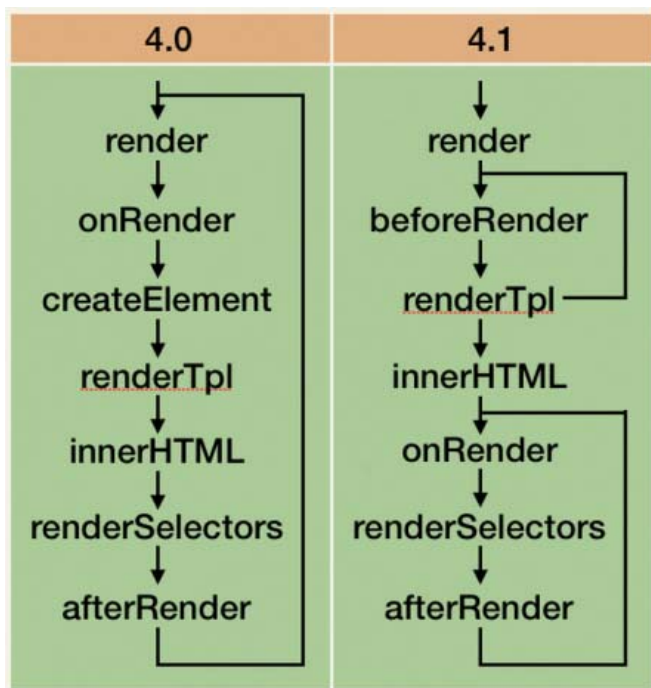
下一步要做的是将这些初始化组件和容器转换HTML。在以前版本的ExtJS，渲染采用的是调用createElement方法和设置innerHTML的混合方式。在ExtJS 4，每个组件的主元素会使用createElement创建，而内部结构则由名称为“renderTpl”的XTemplate实例产生。

如果一个组件是由容器产生的，如面板，附加元素会使用createElement方法创建，其子组件会重复渲染处理过程渲染到面板的主体元素内。在这每一步中，组件的特殊方法会被调用及特殊事件会触发，让派生类或应用进行扩展处理。

在ExtJS 4.1，优化了组件的渲染，会使用批处理方式进行渲染。批处理渲染不再反复调用createElement方法和innerHTML，而是生成整个组件树的HTML，然后一次性的使用innerHTML将其添加到DOM中。

为了实现这种变化，为组件添加了一个名称为“beforeRender”的新方法。虽然一直有beforeRender事件，但派生类如果需要主元素被创建前做一些操作，通常不得不在重写render或onRender方法之间进行选择。它们做它们需要做的事，然后调用基本版本的方法，然后创建元素。

图1展示了4.0和4.1的渲染流程。这两个流程，其处理都开始于一个特定的组件及其内部的组件树。



布局

一旦DOM已经包含了所有必要的元素，最后的步骤就是处理特殊元素的大小和位置。也就是说，最终的步骤是对组件进行布局。这个处理过程相当复杂和费时。在4.0.7的主题示例中，它占了加载总时间的一半以上。布局的挑战主要来自于浏览器处理样式的方式（如margin、width和height），尤其是这些方式一路上都在改变。

性能的第一条规则是CSS计算是昂贵的。正因为如此，浏览器会缓存这些结果。当Javascript在设置宽度或高度时，会让浏览器的缓存部分或全部无效。一个函数会影响多少缓存，依赖于浏览器CSS引擎及其聪明程度。样式信息的下一个请求通常会触发一个reflow过程来刷新缓存。通常来说，“读写=reflow”。鉴于reflow的昂贵，一个明显能提高性能的方法是减少布局的reflow过程。

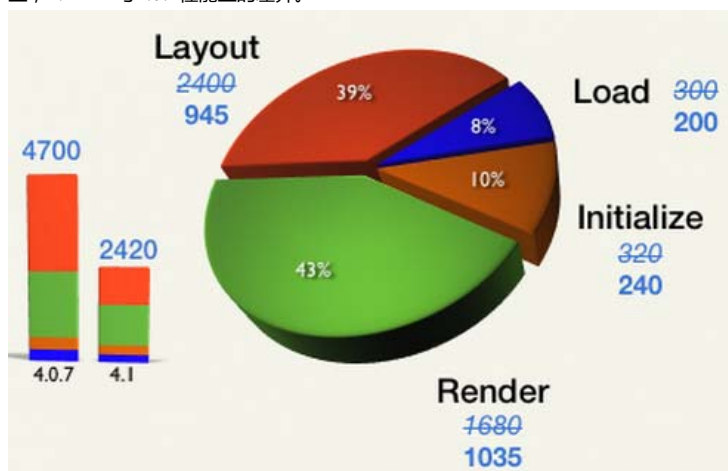
在ExtJS 4.0，一个HBoxLayout的例子，只有在它读取了它需要知道的每一个组件的信息后，才会缓存所有计算和写入这些结果。如果HBoxLayout需要组件的大小，它必须去测量组件元素（read），但是，在做这个之前，组件布局必须先完成它的工作。也就是说，组件布局进行了一些计算（read），然后保存这些结果到DOM（write）。然后，HBoxLayout测量这个组件元素（read）。

这些开始是读取队列，接着是写入队列的操作，往往会高度交错在一起进行读写，这就产生了大量reflow过程。为了消除这些子布局的reflow过程，需要采用外部DOM的方式，将其结果报告给它们的拥有者。

在ExtJS 4.1的已经重构了布局，通过布局上下文对象来共享这些结果以避免DOM的读写（及其相关的reflow过程）。在大量内部使用自定义布局的时候会感觉到这个变化。我们相信这在一个更新版本是相当罕见的做法。

从4.0.7到4.1.0 PR1

所有这些优化都产生了非常显著的收益。用于ExtJS基本性能测试的关键示例之一主题示例。图2展示了在这个示例在IE8上，4.1 PR1与4.0.7性能上的差异。



下一步

显然4.1是4.0一个重大改进，但它的速度还没有超越3.4。而这并不是性能优化的决定性因素。事实上，在4.x，还有许多其它性能优化计划，但并不适合在这版本。现在我们的目标是快速的发布一个稳定的ExtJS 4.1发布版。我们将会努力工作以便在后续版本中提供那些额外的收益。

其它方面

作为承诺，该版本不单纯是与性能有关。今年在SenchaCon展示的海王星主题，其预览版将是本版本的一部分。更令人高兴的是，日历示例将回归。

以下列表可以包含许多其它改进，但我们更想深入一些更令人兴奋的变化。

Grid

应大家需求，看是否有其它解决方案来解决ExtJS 4.0中的缓存滚动条和无限滚动机制。很高兴，事实上我们可以不使用虚拟滚动来解决这个技术问题。

在ExtJS 4.1，几个所有Grid都会使用原生的滚动条。这大大改善了用户的体验，因为，Grid的加速度、滚动量和摩擦量和其它的滚动内容是一样的。另一个可喜的改进是，意味着滚动效果是基于像素而不是整行。这让无限Grid变得更真实，即使行的高度是不同的。

例外的情况是在需要使用虚拟滚动条锁定锁定Grid的一半，因为它没有滚动条，而原生滚动条在这里不可选。

最后，虽然不是Grid的本身一部分，现在可以使用Store处理元数据（metadata）。

BorderLayout

在布局的工作过程中，边界布局在一些内部结构调整中有特别的改进。一直以来，它是非常流行的布局，但也长期受到了一些限制：

- north、south、east或west等区域，只允许一个。如果需要多个south区域，你需要在其内部嵌套边界布局。

- 不能在north或south区域中优先配置east或west区域。要做到这一点，需要使用嵌套边界布局。

- 创建后，不能添加组件到容器。

- 创建后，组件不能从容器中移除。

我们很高兴地说，这些限制在ExtJS 4.1已被删除。（🙄，这改进太给力了）

XTemplate

在内部，ExtJS经常会使用XTemplate类。这是框架极为重要的一部分，但它缺少一些重要特性：它不能有效的在数组的子过程中加入操作。当开始批量渲染工作时，我们决定DomHelper和XTemplate需要协作处理标记生产，推动其输出到一个共享阵列。

然后，我们发现XTemplate内部可以不通过外科手术式的修改，以支持这项工作。这让我们重新考虑这项工作所需。

Xtemplate将会是长期存在的挑战和问题。

- 它只支持最基本的控制结构：“for” 和 “if” 。

- 从模板生成的代码介于很难与不可调试之间。因此，模板文本中的错误很难追查。

- 模板文本会在模板构建时间内被编译，这是不可取的，因为许多模板实例事实上从来没有被使用。

- 执行模板的编译代码并不如想象中的快，因为它包含许多内部函数调用和字符串串联。

现在在ExtJS 4.1，Xtemplate会在第一使用它们的时候才编译。这几乎是免费的Xtemplate构建。此外，编译后的代码可以使用调试器单步调试，而且看起来更象原始模板。

通过这种方法，很多事情变得很容易支持，如“else”、“else if”或“switch”语句。即使在文本中插入代码（类似JSP或ASP）现在也是一个很简单的扩展。

```
01. var tpl = new Ext.XTemplate(  
02.     '<tpl for="orders">'  
03.     'Order {id} is '  
04.     '<tpl if="total > 100">'  
05.         'large',  
06.     '<tpl elseif="total > 25">'  
07.         'medium',  
08.     '<tpl elseif="total > 0">'  
09.         'small',  
10.     '<tpl else>'  
11.         '{% continue; %}',  
12.     '</tpl>'  
13.     'Items:',  
14.     ...  
15.     '</tpl>');
```



“” 语句将产生 “for” 循环，而 “”、“” 和 “” 显然会产生适当的 “if” 和 “else” 块。

新的 “{% x}” 语法类似 “[{x}]” 。这个主体被视为任意代码。在 “[{x}]” 表达式，x是一个表达式，在输出时将产生一个值。而 “{% x}” ，x简单的表示插入一个功能。在当前示例，运行到它时，将继续执行循环。

overrides

在ExtJS，长期以来一直使用“重写”的方式进行错误修复和改进。在过去，这些都必须手动管理这些特殊实体。对现有类的维护，几乎所有其它代码在ExtJS 4.0中会使用类名作为字符串，例如，从Ext.panel.Panel派生：

```
1. Ext.define('My.app.Panel', {  
2.     extend: 'Ext.panel.Panel',  
3.  
4.     method: function () {  
5.         this.callParent();  
6.     }  
7. });
```

但应用一个重写在同一Panel类（在ExtJS 4.0），写法完全不同：

```
1. Ext.panel.Panel.override({  
2.     method: function () {  
3.         this.callOverridden(); // not possible before 4.x  
4.     }  
5. });
```

如果Panel类还没加载，这段代码将会执行失败。继承的使用示例则不会失败，而是告知需要加载/创建Ext.panel.Panel。

重写现在是一等公民。在需要的时候，它们可以被命名和加载。事实上，编写一个重写与写一个派生类相同。

```
1. Ext.define('My.app.PanelPatch', {
2.     override: 'Ext.panel.Panel',
3.
4.     method: function () {
5.         this.callParent();
6.     }
7. });
```



这不仅支持传统的托管方式的创写，也可以成为一个类似mixin的设计工具。mixin类总是作为类的一部分（类似基类），重写可根据预期或需要在后期固定。

小结

我们希望你接近发布ExtJS 4.1最终版的时候有机会下载和尝试新功能及改进。我们也希望收到大家对于该版本的感受与反馈，以及进一步需要需要改善的地方。


英文:

<http://www.sencha.com/blog/whats-new-in-ext-js-4-1/>

译者:黄灯桥 <http://blog.csdn.net/tianxiaode/article/details/7045353>

关键字: 相关新闻, ExtJS 4.1, ExtJS 4, ExtJs

要发表评论，请先[登录](#) 1147 次阅读



辽宁金禾实业有限公司

0415-2808711

**专业生产：HPS高性能
特殊单立管**