


[Обучение Ext JS] :: Часть 10. Эффекты в ExtJS

Мар 14 2010 

Самое простое, что можно сделать при создании приложения, это переключиться в режим программирования с целью сосредоточиться на коде, а не удобстве пользователя. Архитектура системы очень важна, но если пользователю не понравится взаимодействовать с ней, то проект может закончиться провалом.

Ext JS предлагает ряд удобных способов решения этой проблемы. Элементы удобны, красивы и отлично функционируют во всем оформлении.

В большинство из компонентов **Ext JS** по умолчанию уже встроена переходная анимация, позволяющая гладко разворачивать узел дерева или уменьшать окно до размеров кнопки, которую можно активировать позднее.

Подобные особенности это не что-то вроде вишенки на пироге, нет, они идут из глубин и встроены на программном уровне для удобства как программистов, так и конечных пользователей.

Все примитивно

В основе большинства особенностей **Ext JS** лежит `Ext.Element`, и в самом деле метод `Ext.Component.getEl` отражает тот факт, что такие виджеты как окна полей формы и панели инструментов поддерживаются этим блоком.

В `Ext.Element` смешивается большинство методов из класса `Ext.Fx`, созданного для гладких переходов и анимации для элементов **HTML** и схожих компонентов. Мы будем обсуждать увлекательные возможности, открывающиеся нам при использовании `Ext.Fx` provides, а также обсудим как это работает совместно с `Ext.Element`.

Забавные особенности

Существуют несколько классов, предоставляющих дополнительную функциональность для улучшения взаимодействия с пользователями. Большинство из этих особенностей после установки не требуют дополнительной настройки и сразу готовы к использованию. Мы, также, можем использовать их по отдельности для создания наполненных подсказок, скрытых предметов, которые еще только поджужаются или для выделения отдельных частей экрана, привлекая внимание пользователя.

It's ok to love

На самом деле, большинство функций, которые мы рассмотрим в этой главе по своей природе поверхностны. Они добавляют гладкие переходы... но и без них можно обойтись. В их защиту можно привести два аргумента. Во-первых, хороший разработчик понимает, что пользователи далеко не роботы. Во-вторых, и что более важно, переходы необходимы для того, чтобы показать что на странице что-то случилось. Резкое изменение картинки обычно приводит к реакции вида "Это что сейчас было?". переходы привлекают внимание пользователей и помогают осознать изменения.

Fx-ичные функции

`Ext.Fx` представляет собой отдельный класс, архитектурно независимый от `Ext.Element`, но в жизни он не может использоваться самостоятельно. Вместо этого, он автоматически встраивается в методы и свойства класса `Ext.Element`. Таким образом каждое действие из `Ext.Fx` доступно для каждого случая `Ext.Element`. Ниже мы обязательно приведем примеры этого.

Методическое сумасшествие

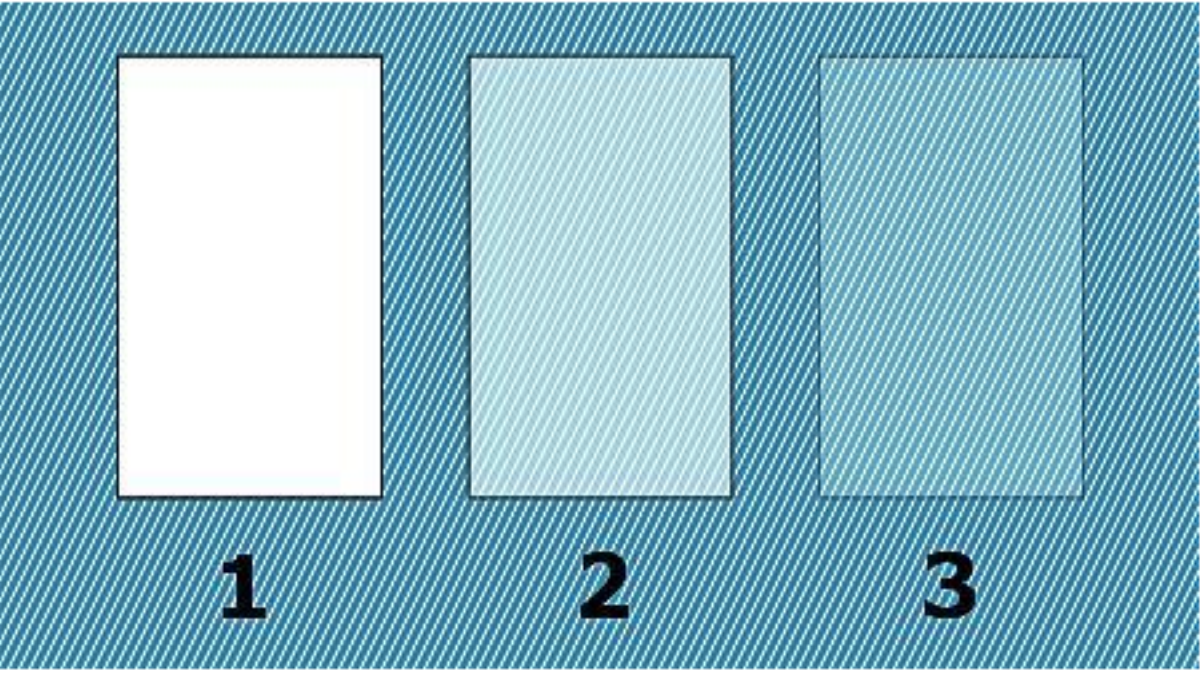
Как уже упоминалось ранее, у `Ext.Fx` имеется набор методов, вызываемых для выполнения некоторой магии. Каждый их этих методов доступен для случаев `Ext.Element`. Таким образом в последующих примерах мы сначала убедимся, что на вашей **HTML** странице есть элемент `div` с ID "мишени".

Затемнение

Термин "затемнения" применяется в `Ext.Fx` по отношению к изменению прозрачности — от 100% ясности до 0%(ослабление) и наоборот (усиление). Другими словами мы заставляем что-то исчезнуть, и снова появиться, но так как наша глава посвящена эффектам, этот переход анимирован. Два метода, которые выполняют эти переходы: `Ext.Fx.fadeOut` и `Ext.Fx.fadeIn`. Оба поддерживают простое использование, без аргументов:

```
Ext.get('target').fadeOut();
window.setTimeout(function() {
    Ext.get('target').fadeIn();
}, 3000);
```

Это заставит целевой элемент медленно исчезать. По истечении 3-х секундной задержки он так же медленно появится.



Можно настроить поведение этих методов, используя опцию настройкой `endOpacity`. А теории это позволяет указывать какой элемент не будет исчезать полностью. Полезно, если вы хотите указать на то, что элемент больше не представляет важности. Делается это примерно так, и при этом элемент не вернет своей изначальной "плотности":

```
Ext.get('target').fadeOut({endOpacity:0.25});
window.setTimeout(function() {
    Ext.get('target').fadeIn({endOpacity: 0.75});
}, 3000);
```

Обратите внимание, я сказал "в теории". На самом деле, ограничения в **Ext JS 2.2** означают, что наш предмет просто исчезнет при достижении `endOpacity`. Я надеюсь что в следующей версии это исправят.

В вышеприведенном примере, изначально-плотный предмет должен становиться прозрачной на четверть прежде чем вернуться к показателю в три четверти через три секунды.

В добавление к опции настройки, такой как `endOpacity`, являющейся особенной для методов индивидуальных эффектов, существует несколько общих опций, которые присущи всем методам `Ext.Fx`. Это позволяет настраивать поведение эффектов. Чуть позже мы это обсудим.

Кадрирование

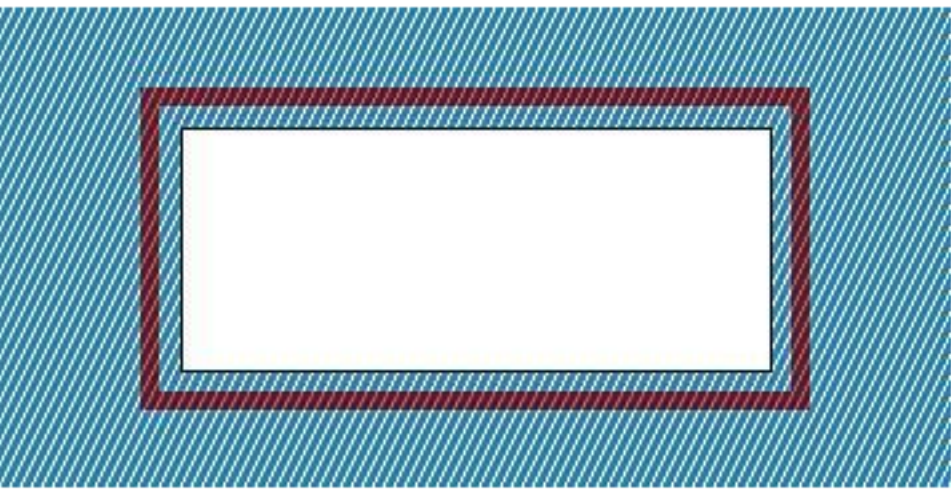
Кадрирование напоминает импульс радара из видеоигры. Оно испускается из начальной точки, и постепенно исчезает. Главная область применения - выделить какой-либо элемент экрана. Методы `Ext.Fx.frame` также позволяют убедиться, что внимание пользователя будет надежно привлечено многократными "импульсами". Самый простой способ использования без аргументов:

```
Ext.get('target').frame();
```

Это заставляет целевой элемент испускать единичный светло-голубой импульс, привлекающий внимание к, собственно, элементу. Тем не менее, для того, чтобы убедиться в том, что пользователь знает о более важном событии, можно использовать что-то вроде этого:

```
Ext.get('target').frame('#0000', 3);
```

Первый аргумент это шестнадцатеричный код цвета эффекта. В нашем случае это агрессивно-красный. Второй аргумент указывает количество импульсов. Таким образом мы используем три красных импульса для обозначения того, что сейчас произойдет что-то очень плохое.

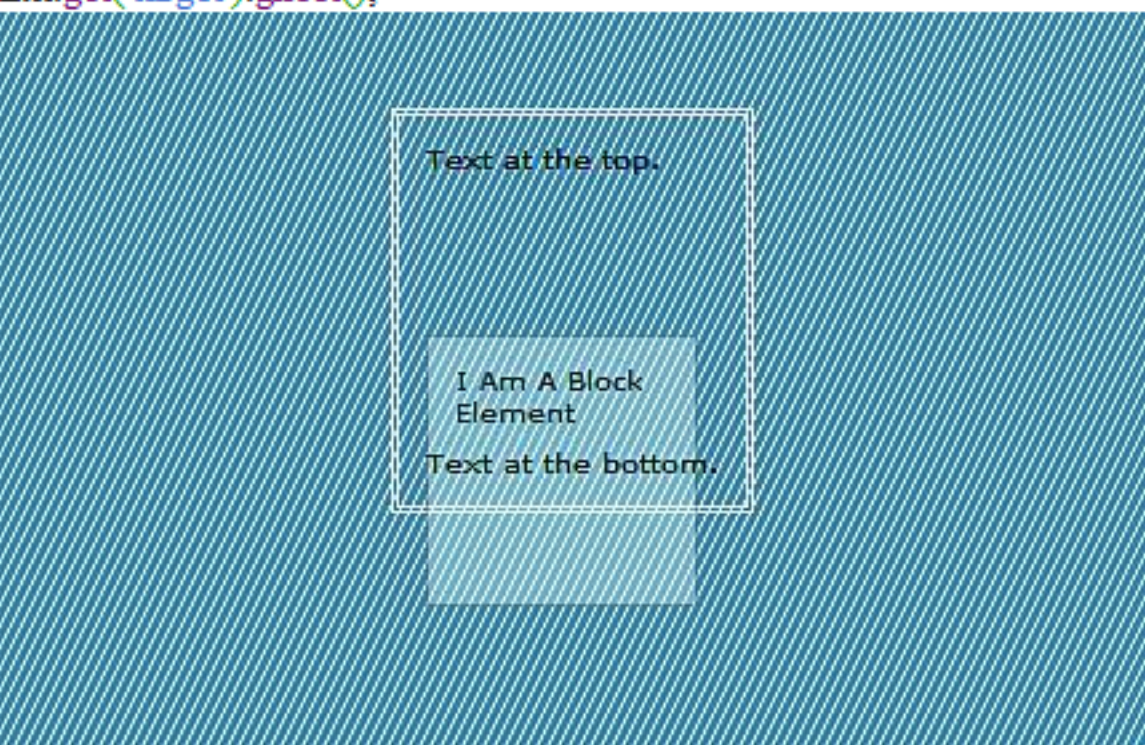


Вот она настоящая сила этого метода: повторение, и возможность использовать различные цвета для отображения разнообразных ситуаций и приоритетов.

Бу! Призраки.

Призраки - или точнее, появление ореола, это термин, которым **Ext JS** называет исчезновение элементов, пока они переходят в другое место. Этот эффект используется для создания впечатления, что элемент перешел из одной области в другую, например, задействование раскраковки через несколько изображений может включить призрака, если выбрано предыдущее звено. По умолчанию призрак заставляет элементы падать во время исчезновения.

```
Ext.get('target').ghost();
```



Установки по умолчанию можно использовать для избавления более ненужных от элементов. Возможно удаление событий может быть значительней, при обращении к `Ext.Fx.ghost`. Как уже упоминалось ранее, можно также обслуживать другие ситуации, указывая другое направление для призрака в зависимости от обстоятельств. Стандартный якорь `Ext.Fx` используется в анимации и выравнивании так же может быть использован совместно с методом призрака:

```
Ext.get('target').ghost('tr');
```

Здесь мы указали, что цель будет перемещаться в направлении своего верхнего-правого угла, но можно задействовать любой доступный якорь. Позже мы поговорим про якоря подробнее. Но это является тем параметром, который позволяет копировать функционал кадрирования. Используя левые и правые якорные точки можно сдвигать призрак в соответствующем направлении.

Выделение

Если вам знаком термин "**Web 2.0**", то вы наверное уже слышали о "технике желтого затемнения". Быстрый веб-поиск превращает прибавляет бесчисленные ссылки к этой особенности **JavaScript**, ставшей популярной когда **Web 2.0** впервые получил известность. Обычно используется в тех случаях, когда действие было завершено, и пользователь был перенаправлен на другую страницу. Этот эффект окрашивается цветом выделения считается дружелюбно-желтый, но для ошибок надо выбрать что-то посильнее. Сказав что желтый, это цвет по умолчанию для `Ext.Fx.highlight`:

```
Looks like I'm highlighted in good ol' yellow...
```

Запускается следующим кодом:

```
Ext.get('target').highlight();
```

Ext JS выводит выделение на следующий уровень. В то время как стандартное использование влияет на цвет фона **CSS** целевого элемента, можно выбрать свойства **CSS**, такие как цвет, и цвет границ. Также можно указать `endColor` выделения. В нашем втором, и более сложном, примере мы заставим текст начать с белого, и закончить зеленым:

```
Ext.get('target').highlight('#fff',{
    attr: 'color',
    endColor: '00ff00'
});
```

Это очень гибкий метод, используемый для выделения больших частей страницы — возможно `fieldset` в форме, или просто блок текста —, например в проверке орфографии онлайн. Отмечу, что можно заставить его реагировать и отвечать на действия пользователя. А гибкость использовать для наглядного обозначения состояния нашей системы.

- 1
- 2
- 3
- [следующая >](#)
- [последняя >>](#)