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

Мар 14 2010**Разделение на блоки и утилиты Ext.Fx**
Существует несколько дополнительных методов, которые не отвечают за создание объектов напрямую. Вместо этого они могут быть использованы для управления и отслеживания эффектов которые уже в очереди. Отличным примером этого является метод stopFx, который очень прост — мы вызываем его для отмены всей запущенной анимации. Он также очищает очередь эффектов и подготавливает элемент для взаимодействия с пользователем и дальнейших манипуляций.
Интересно, что stopFx возвращает целевой Ext.Element, позволяя разработчику сцепить последующие методы. Возможно вы установили slideIn, который последовательно отменен и заменен на slideOut:
Ext.get('target').stopFx().slideOut();
Такой код полезен если нам необходимо прервать анимацию и вернуть управление пользователю — возможность которая может быть при обеспечении пользователю требуемую степень отзывчивости интерфейса. Тем не менее в других обстоятельствах, вы можете захотеть запретить пользователю вмешиваться в текущее состояние приложения, и в таком случае у нас есть метод thehasFxBlock. Обычно это применяется для того, чтобы пользователь не мог создавать очередей из большого количества эффектов. Это осуществимо выставлением возврата элемента ifhasFxBlock на true.

Основы
Как мы уже видели Ext.Element является ключом к использованию Ext.Fx. Важно то, что когда запускаете свою версию **Ext JS**, то следует убедиться. что вы включили все желаемые Ext.Fx. Тем не менее, даже используясь отдельно, у Ext.Element существует ряд методов, которые могут предоставить интересные возможности для анимации.

Совершая движения
Существует большое количество методов, используемое для управления целевым элементом, но у некоторых из них есть бесплатные дополнения. Отправьте значение true в качестве последнего параметра и управление будет анимировано.
Ext.get('target').moveTo(300, 500, **true**);
В этом случае элемент будет перемещен на 300 пикселей от левого края, и на 500 от верхнего. Но поскольку последний параметр не просто прыжок на место, у нас будет мягкий переход.
в дополнение к простому отправлению true как третий параметр мы можем отправить весь объект настройки. И это будет использовать ту же опцию анимации для Ext.Fx что мы обсуждали ранее.

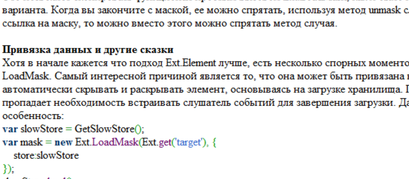
ВАЖНО: *Анимация Ext.Element может быть использована одновременно как и Ext.Fx, но никто не гарантирует, что им всегда будут доступны функции создания очереди.*

Существует много методов, такие как setOpacity и прокрутка, используемые для добавления простых переходов. Это то самое хорошее основание на которое стоит Ext.Fx.
Мы также рассмотрим и другие детали анимационной головоломки:
Ext.Element.animate. Это общее средство применяемой к элементу анимации, и ссылка на основной метод используемый всеми методами Ext.Element которые поддерживают эту анимацию.
Улучшенные применения этого метода применяют записи Ext.lib.anim утверждающие, что весь запас типов анимации доступен для **Ext JS**, и добавляет новые зарегестрированные обработчики для поддержки пользовательских сценариев. В большинстве случаев использование запаса методов Ext.Element или методов Ext.Fx будет хватать, даже для улучшенного класса.

Использование компонентов Ext
Анимация доступная через Ext.Element в особенности интересна когда у вас большинство Ext.Components, таких как Window или FormPanel, имеют основной элемент как контейнер. Это значит, что вы можете применять эти эффекты для полных компонентов, точно так же как и для стандартных элементов, предоставляя методам дополнительные возможности.

Принесите вспышку
Мы рассмотрели Ext.Fx и другие анимационные техники предлагаемые **Ext JS**. Но все еще остается много особенностей которые выходят за рамки основы. Существуют несколько классов **Ext** помогающие добавить немного лоска стандартным решениям. У нас есть LoadMask позволяющая скрывать часть экрана во то время как они обновляется, и QuickTip предлагающая богатую и настраиваемую систему подсказок.
Эти классы открывают перед нами новые горизонты, когда речь заходит о создании очень интересного для пользователя приложения. И дальше мы рассмотрим как это сделать. Ключевая особенность заключается в том, что они (классы) очень просты в употреблении, а это очень весский довод для использования их в приложениях **Ext JS**.

Я так говорю
У класса Ext.LoadMask в использовании существует пара отличий. Мы можем, просто выводить сообщение, или же ограничить, с его помощью, взаимодействие с обновляемым компонентом.



Давайте рассмотрим первый случай. Использование Ext.LoadMask для вывода текстового сообщения поверх элемента.
var target = Ext.get('target');
var mask = new Ext.LoadMask(target);
mask.show();
В вышеприведенном примере, как только мы, используя Ext.get получили элемент, мы переправляем его в конструктор Ext.LoadMask и вызываем метод show для случая LoadMask. Также, для конструктора Ext.LoadMask существуют дополнительные, опциональные параметры: объект настройки с 4 свойствами:

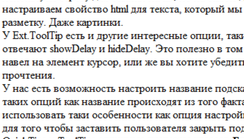
Название свойства	Описание
msg	Сообщение о загрузке, которое будет отображено в центре маски
msgCls	Класс CSS используемый для загрузки контейнера сообщения
removeMask	Булево, установив True уничтожит маску по завершении загрузки
store	Обсудим позже

Создавая маску таким образом, мы можем убрать ее вызвав метод mask.hide. Хотя такой подход довольно прямолинеен, он также многословен. Таким образом в некоторых классах лучше использовать метод ссылки который предоставляемый Ext.Element:
Ext.get('target').mask('Loading...', 'x-mask-loading');
В этом примере мы эксплицитно передаем текстовое сообщение и класс сообщения **CSS** в первом и втором аргументе соответственно. Это позволяет скопировать функционал простым вызовом Ext.LoadMask, как и было показано в первом примере. Но возможны оба варианта. Когда вы закончите с маской, ее можно спрятать, используя метод unmask случая Ext.Element. Если у вас есть сохраненная ссылка на маску, то можно вместо этого можно спрятать метод случая.

Привязка данных и другие сказки
Хотя в начале кажется что подход Ext.Element лучше, есть несколько спорных моментов, подталкивающие к использованию полной LoadMask. Самый интересной причиной является то, что она может быть привязана к указанному случаю Ext.data.Store, позволяя автоматически скрывать и раскрывать элемент, основываясь на загрузке хранилища. При показе маски подрезается код, но также пропадает необходимость встраивать слушатель событий для завершения загрузки. Давайте посмотрим как можно использовать эту особенность:
```
var slowStore = GetSlowStore();
var mask = new Ext.LoadMask(Ext.get('target'), {
   store:slowStore
});
slowStore.load();
```
Мы не заинтересованы в создании хранилища, таким образом, оно было скрыто под вызовом функции. Сейчас мы беспокоимся о том, чтобы наше учебное хранилище через несколько секунд закончит загружаться. мы запускаем новую LoadMask с элементом в качестве первого параметра конструктора, и буквенной настройкой в качестве второго. Свойство хранилища принимает slowStore в качестве значения.
Теперь, когда все настроено, мы просто вызываем метод из хранилища. Это скроет нам элемент и через несколько секунд, когда загрузка завершится, маска исчезнет.

Учитывая компоненты
Как мы уже видели, существует несколько способов сокрытия элементов. В Ext.Elements можно использовать большинство из этих подходов, ведь он является основой для большинства Ext.Components. Тем не менее, важно заметить, что некоторые компоненты предоставляют больше функционала, сокращая количество кода, который вам в ином случае пришлось бы писать. Например у Ext.grid.GridPanel существует логическая опция настройки loadMask, которая не только скрывает панель, но и автоматически привязывается к хранилищу данных, которое обеспечивает GridPanel.

QuickTipping
Часто бывает полезно иметь возможность дать о кнопке или поле формы дополнительную информацию. Просто на случай если пользователю будет непонятна ее функция. В большинстве случаев, размещать эту информацию инлайн, рядом с проблемным элементом, довольно непрактично. Как минимум потому, что это загромоздит интерфейс. В такой ситуации всплывающая подсказка является отличной заменой, а в **Ext JS** эти маленькие всплывающие штучки могут содержать любой **HTML**.
Класс Ext.QuickTips является одноэлементным и предоставляет очень простые методы для создания богатых всплывающих подсказок. Он позволяет размещать информацию по настройке QuickTip непосредственно в разметку страницы. В большинстве случаев имеет смысл текст описания в пределах главного **HTML**. Этот подход поддерживается при помощи дополнительных атрибутов для любого элемента **HTML** к которому вы захотите привязать QuickTip. Чаще всего используют ext.qwidth, ext.qtitle и ext.qtip:
```
<input type="text" ext:qtitle="Information" ext:qtip=
        "This field can contain text!" ext:qwidth="200" />
```
Нам нужно запустить глобальный обработчик QuickTips прежде чем он начнет работать:
Ext.QuickTips.init();
Также, стоит упомянуть, что могут вызвать нестандартные атрибуты после выполнения двух этих шагов. У вас получится вот такая подсказка:



Так же как и подсказки встраиваемые в разметку, при помощи класса Ext.ToolTip можно программно настраивать их. Это дает выбор поций настройки, для изменения поведения и внешности подсказок, а также позволяет вытаскивать данные из внешних источников используя возможности **AJAX**.
Самый простой пример использования этого класса:
```
new Ext.ToolTip({
   target: 'tipTarget',
   html: 'This is where our information goes!'
});
```
Конструктор для Ext.ToolTip принимает буквенные настройки в качестве единственного аргумента. И здесь мы используем обязательное свойство цели для указания какой элемент запустит подсказку. В этом случае это будет то, что с ID tipTarget. Обратите внимание, что мы используем строку для ссылки на ID, но на самом деле можно передать в виде ссылки Ext.Element или узлом **HTML**. Затем мы настраиваем свойство html для текста, который мы хотим показать в подсказке. Как гласит название свойства, можно добавить любую разметку. Даже картинки.
У Ext.ToolTip есть и другие интересные опции, такие как возможность настроить время, за которое появится и исчезнет подсказка. за это отвечают showDelay и hideDelay. Это полезно в том случае, когда вы хотите чтобы подсказка появилась сразу же как только пользователь навел на элемент курсор, или же вы хотите убедиться что информация будет показываться достаточное время, которого хватит для прочтения.
У нас есть возможность настроить название подсказки точно тем же способом что мы делали и в разметке. На самом деле, большинство таких опций как название происходят из того факта, что Ext.ToolTip является подклассом Ext.Panel. Это значит, что мы можем использовать таки особенности как опция настройки autoLoad для захвата содержимого из URL, или использовать закрываемую опцию для того чтобы заставить пользователя закрыть подсказку при помощи виджета.
QuickTips и ToolTips отличные примеры классов **Ext JS** которые не жизненно нужны приложению, но являются слоем функционала который переносит некоторые решения из разряда обычных в очень полезные..

Заключени
Существует четкая линия между визуальными знаками и раздражающими эффектами. Хотя мы рассмотрели много особенностей **Ext JS** позволяющие сдвинуть приложение в любую из крайностей, эта глава была только об инструментах, а не том как вы должны их использовать.
Также мы рассмотрели пару примеров, которые объяснили насколько полезно и красиво помогать пользователям. Предоставляя нечто большее при удалении записи или закрытии экрана с данными вы превращаете возможность запутаться в плавные и понятные переходы.
Главное убедиться что анимация не слишком постепенная, и что она не превышает допустимые нормы. И что не запускается всякий раз когда пользователь нажимает кнопку. Софтверные приложения просто работают и им красивости не очень нужны.
Хотя **Ext JS** не задумывался ради таких эффектов. Мы показали пару классов являющиеся классами утилит стиля. ToolTips и LoadMasks невероятно полезно в большинстве случаев и редко могут быть использованы в избытке..
ToolTips могут добавить много ценного в загруженное приложение, удаляя беспорядок, смещая инлайн текст в всплывающие окна и предоставляя единую и привлекательную помощь. Изображения и даже **HTML** ссылки могут дать больше информации чем то, что расположено коло формы поля.
Второй из рассмотренных нами классов, LoadMask, может образовывать важные части системы, удостоверяясь не только в том, что пользователи не прервут загрузку элемента, но и демонстрирует то, что маска может быть привлекательной, содержательной и хранить информацию которая может заинтересовать пользователя.
В следующей главе мы рассмотрим один из наиболее типичных примеров блеска **Web 2.0**: drag-and-drop. В типичной для **Ext JS** манере мы увидим не только простоту использования, но и силу функциональности

- « первая
- ‹ предыдущая
- 1
- 2
- 3

English

Голос