MANUAL DO EXT CORE

TRADUÇÃO LIVRE

Traduzido por: Otávio Augusto Rodrigues Fernandes

<otavio@neton.com.br>

Visão panorâmica de Ext Core

Introdução

Ext Core é uma biblioteca JavaScript leve e com recursos ricos disponível sob uma licença MIT. Ext core é um pacote cheio de recursos emocionantes projetados para permitir o desenvolvimento web rápido, enquanto encoraja um código bem projetado e escalável. Esta biblioteca fornece abstrações para a manipulação de DOM, Ajax, Eventos e Eventos customizáveis, animações, templates, mecanismos OO e mais. Ext Core é liberada sobre uma licença MIT e é perfeita para a inclusão em uma página web dinâmica ou mesmo pequenas aplicações.

Download

Ext Core pode ser baixada aqui e as últimas informações podem ser encontradas na Pagina do Projeto Ext Core.

Incluindo Ext Core

O pacote de Ext Core vem uma versão para debug e com uma versão de produção. A versão de produção foi minimizada (espaços em branco, retorno de linha e comentários removidos) e obfuscados (todas as variáveis locais foram renomeadas para versões menores) usando YUI Compressor. Você deve sempre usar a versão debug quando estiver desenvolvendo, uma vez que você irá receber mensagens de erro(não obfuscadas) mais informativas.

Para incluir a versão de desenvolvimento da biblioteca Ext Core simplesmente inclua o seguinte arquivo JavaScript:

```
<script src="ext-core-debug.js"></script>
```

E para a versão de produção (25k comprimido e obfuscado), simplesmente remova "-debug":

```
<script src="ext-core.js"></script>
```

É isto. Não há arquivo css para incluir com Ext Core.

Exemplo simples

Depois que você incluiu Ext Core em sua página web, experimente o seguinte código para verificar se tudo está funcionando corretamente.

```
Ext.onReady(function() {

    Ext.DomHelper.append(document.body,   {tag:   'p',   cls:   'some-
class'});

    Ext.select('p.some-class').update('Ext  Core  foi  adicionada  com
sucesso');

});
```

Sobre

O manual do Ext Core é autoria de Tommy Maintz, Aaron Conran, James Donaghue, Jamie Avins e Evan Trimboli. Obrigado a todos vocês revisores que revisaram o manual antes desta liberação. No momento nós estamos a estudar traduções para todos os outros idiomas.

Ext.Element

Compreendendo Ext.Elements

Um documento HTML tipicamente consiste de um grande número de marcações HTML. Quando o navegador carrega seu documento HTML, cada tag em sua marcação é traduzia em um HTMLElement e o navegador constrói uma árvore de marcações Document Object Model (DOM). Essa árvore DOM é armazenada no escopo global do navegador em uma variável chamada *document*. Esta variável document detém uma referência a cada parte da marcação que tenha sido carregada na página.

O objeto *document* fornece um método importante chamado *getElementById*. Este método permite que você recupere o HTMLElement subjacente de qualquer navegador. Entretanto, você deve estar ciente de que há muitas diferenças entre navegadores quando esses manipulam diretamente o DOM. Ext Core implementa uma classe chamada Ext.Element que encapsula o HTMLElement típico, fornecendo a você um suporte cross-browser.

Ext.Element possui a maioria dos métodos de qualquer classe na biblioteca Ext Core. Esses métodos podem ser categorizados como se segue:

CSS e Estilização

(setStyle, addClass)

Dom Querying ou Navegação

(query, select, findParent)

Manipulação Dom

(createChild, remove)

Dimensões

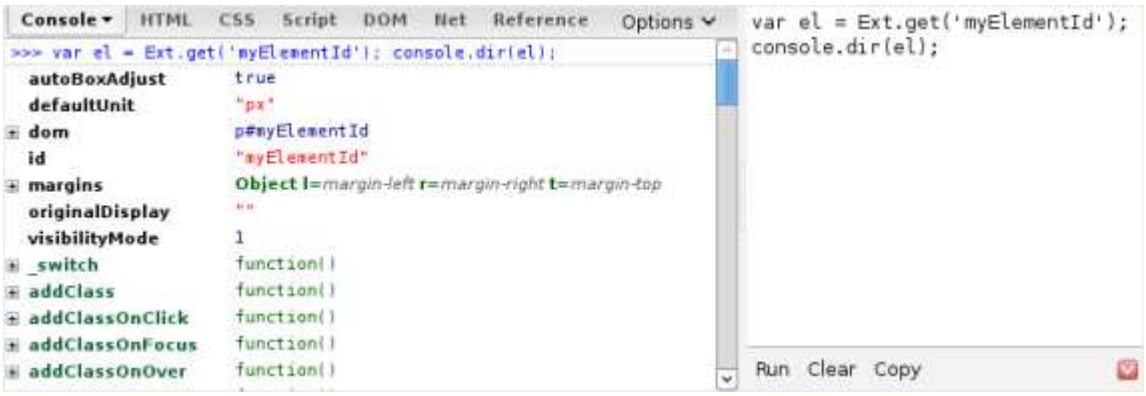
(getHeight, getWidth)

Você pode usar o método Ext.get para criar uma instância de um Ext.Element que encapsula qualquer HTMLElement. Por exemplo, se sua marcação tem um elemento com um id 'myElementId' você pode recuperá-lo desta forma.

```
var el = Ext.get('myElementId');
```

Usando o Firebug, tente executar o seguinte código. Observe os métodos expostos pelo Ext.Element. Lembre-se de que por você estar olhando o objeto JS nativo, ambos os métodos privados e públicos estão visíveis. Em caso de dúvida, consulte a documentação da API.

```
var el = Ext.get('myElementId');

console.dir(el);
```



console.dir é um método fornecido pelo Firebug que irá dar um dump no conteúdo do objeto em um formato facilmente visualizável. Ele também permite que você expanda e esconda sub-objetos que possuem propriedades adicionais. Propriedades são mostradas na cor preta. Métodos/Funções são verdes; construtores ou classes em vermelho.

Agora iremos tentar interagir com o parágrafo acima que tem o id *myElementId.*

```
var el = Ext.get('myElementId');

el.addClass('error');
```

Depois de executar este código, a cor da fonte no parágrafo acima será mudada para vermelho. Esta página tem uma regra CSS que modifica todos os elementos com a classe error para a cor vermelha. A regra CSS seria como esta:

```
.error {

        color: red;

}
```

A próxima seção em Ext.Element (Classes CSS & Estilização) irá cobrir muitas maneiras de interagir com classes CSS e estilização de seus elementos.

Introduzindo o Flyweight

O padrão Flyweight Design é um padrão utilizado para minimizar o uso de memória através da criação de um objeto simples global e então re-utilizando este objeto uma e outra vez novamente. Ext cria um Ext.Element global para usá-lo como Flyweight quando ele se inicia. Esta instância pode ser usada para atingir qualquer nó na DOM. Para acessar este objeto flyweight você pode usar o método Ext.fly. Os novos em Ext frequentemente expressam confusão em usar Ext.get ou Ext.fly:

Sempre que uma referência a Ext.Element precisa estar salva para uso futuro, use o método Ext.get. Para vezes em que você não precisa armazenar uma referência ao elemento, você pode usar o flyweight Ext.element compartilhado em toda a biblioteca. Para acessar esse objeto flyweight use o método Ext.fly(elementId).

Vamos remover a classe error do parágrafo acima.

```
Ext.fly('myElementId').removeClass('error');
```

Quando este código é executado, Ext foi capaz de re-utilizar o objeto flyweight compartilhado e não tem que instanciar uma nova instância de um Ext.Element. O método fly é apropriado em situações de execução de uma operação atômica simples de uma única linha. Você nunca irá querer tentar armazenar uma referência do objeto flyweight, uma vez que este está sujeito a mudanças por alguma outra parte de código. Por exemplo, vamos dar uma olhada no seguinte código exemplo.

```
var el = Ext.fly('foo');

Ext.fly('bar').frame();

el.addClass('error');
```

frame é um efeito de highlighting que está incluído no pacote de animação de Ext.Element O que você acha que seria o resultado deste código?

A resposta é que o elemento com um id 'bar' terá o efeito frame aplicado a ele e então, imediatamente após, irá ter a classe CSS error aplicada a ele. Nada irá acontecer ao elemento com o id foo, porque a referência El aponta para o flyweight global que foi sobrescrito quando nos usamos o efeito frame em bar. Se você não expera por isso, releia a seção sobre flyweights uma vez que ela é um conceito importante se você quiser usar o método Ext.fly.

Ext.get

Ext.get recebe uma string id do HTMLElement, Ext.Element, como parâmetro e retorna uma nova instância de um Ext.Element.

Todos os três exemplos a seguir retornam um Ext.Element:

```
var el1 = Ext.get('elId'); // pega um elemento pelo id

var el2 = Ext.get(el1); // pega um elemento pelo Ext.Element

var el3 = Ext.get(el1.dom); // pega pelo HTMLElement
```

Ext.fly

Ext fly recebe o mesmo argumento que Ext.get, mas retorna uma referência para um flyweight gerenciado internamente, instância de um Ext.Element. Nunca armazene uma referência ao objeto flyweight visto que ele irá mudar pela invocação deste método. Ele é projetado meramente como um mecanismo para melhorar a performance, por contornar a necessidade de instanciar um novo Ext.Element quando uma referência não é necessária(se você quer executar uma ação).

Aqui está um exemplo do uso de Ext.fly

```
 // Nós queremos executar somente uma discreta ação neste elemento.
Ext.fly('elId').hide();
```

Ext.getDom

Ext.getDom retorna um nó dom para o String (id), nó dom, ou Ext.Element.

Aqui estão alguns exemplos:

```
// pega o nó dom baseado no id

var elDom = Ext.getDom('elId');

// pega o nó dom baseado no nó dom

var elDom1 = Ext.getDom(elDom);

// Se você não sabe se nós estamos trabalhando com um

// Ext.Element ou um nó dom use Ext.getDom

function(el){  var dom = Ext.getDom(el);

 // faz alguma coisa com o nó dom

}
```

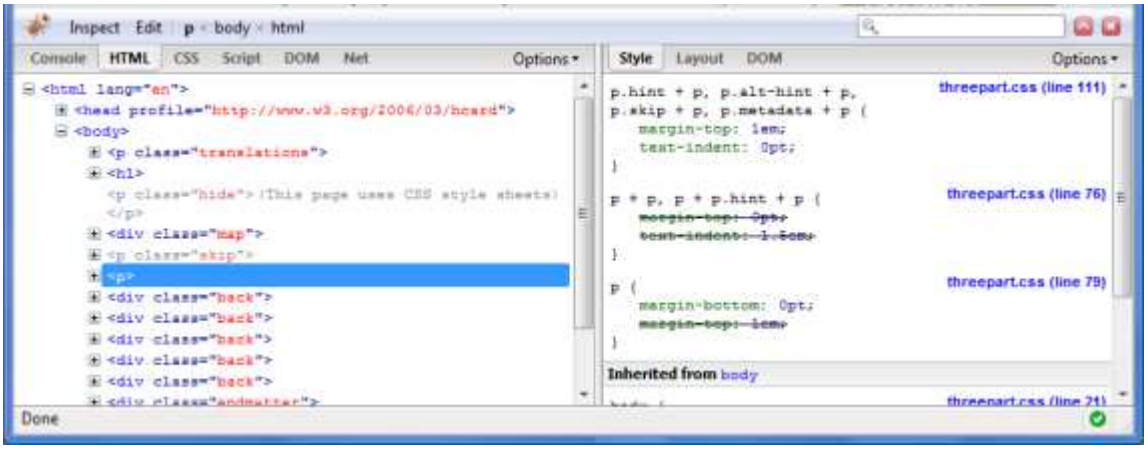
Classes CSS e Estilização

Nós temos aprendido sobre marcações, como elas se relacionam com o documento e os métodos que Ext Core fornece tornando fácil recuperar esses dados. E, no entanto, sobre o layout do documento? Como nós manipulamos o layout e estilização do documento? A resposta é usar a estilização da Cascading Style Sheets (CSS). CSS é uma linguagem pela qual nós podemos controlar o layout e a informação visual de nossa página. Ext Core torna isto realmente fácil, manipular o estilo dos elementos no documento através de classes ou pela modificação de estilos diretamente.

```
<style type="text/css">

myCls {

        color: #F00;

}

</style>


...


<div type="myCls">Olá</div>
```

No exemplo acima a classe "myCls" é aplicada na div dando ao texto "Olá" uma cor visual de vermelho (#FF0).

Agora que nós já temos aprendido sobre o firebug, dê uma olhada em algo mais da ferramenta maravilhosa que ele nos oferece. Clique direito em qualquer elemento na página e escolha a opção "Inspect Element". Agora no firebug nós podemos ver a árvore dom e onde este elemento está localizado. Se você olhar para a direita desta árvore irá ver um painel com tudo do estilo aplicado a este elemento.



Se você ainda não está familiarizado com o uso do FireBug, dê uma pausa agora e aprenda sobre ele – ele é o osciloscópio do desenvolvimento web. Inspect-element vem a calhar para alterar estilos em um site existente ou criar e debugar estilos em um novo site. Assim, vamos explorar os métodos que Ext Core fornece para tornar as modificações css mais fácil pra você.

addClass

Para adicionar fácilmente uma classe a um elemento:

```
Ext.fly('elId').addClass('myCls'); // adiciona a classe 'myCls' ao elemento
```

radioClass

Adiciona uma classe em um elemento e remove a mesma classe, se ela existir, de todos os seus irmãos em uma chamada

```
// adiciona a classe 'myCls' ao elemento e remove a mesma classe de todos os seus irmãos

Ext.fly('elId').radioClass('myCls');
```

removeClass

Remove uma ou mais classes de um elemento.

```
Ext.fly('elId').removeClass('myCls'); // remove a classe do elemento
```

toggleClass

Troca uma classe on/off. Se ela não está aplicada ao elemento ela será aplicada, se ela estiver, ela será então removida.

```
Ext.fly('elId').toggleClass('myCls'); //  a classe é adicionada

Ext.fly('elId').toggleClass('myCls'); //  a classe é removida

Ext.fly('elId').toggleClass('myCls'); // a classe é adicionada de novo
```

hasClass

Verifica se o elemento possui uma classe aplicada.

```
if (Ext.fly('elId').hasClass('myCls')) {

        //  ele tem a classe

}
```

replaceClass

Substitui uma classe **a** por uma classe **b**.

```
Ext.fly('elId').replaceClass('myClsA', 'myClsB');
```

getStyle

Pega uma propriedade normalizada (currentStyle e estilo computado) do elemento.

```
var color = Ext.fly('elId').getStyle('color');

var zIndx = Ext.fly('elId').getStyle('z-index');

var fntFmly = Ext.fly('elId').getStyle('font-family');

// ... etc.
```

setStyle

Define a propriedade estilo de um elemento. Recebe uma string ou um objeto literal.

```
Ext.fly('elId').setStyle('color', '#FFFFFF');

Ext.fly('elId').setStyle('z-index', 10);

Ext.fly('elId').setStyle({

        display : 'block',
```

```
        overflow : 'hidden',
        cursor : 'pointer'
});
// anima a transição de cor
Ext.fly('elId').setStyle('color', '#FFFFFF', true);
// anima a transição de cor com uma duração de .75 segundos
Ext.fly('elId').setStyle('color', '#FFFFFF', {duration: .75});
// ... etc.
```

getColor

Pega o valor de uma cor normalizada (6 dígitos hexadecimais)para a propriedade passada, aceita um um valor default se a propriedade não está setada e um prefixo (# é o default).

```
Ext.fly('elId').getColor('background-color');
Ext.fly('elId').getColor('color');
Ext.fly('elId').getColor('border-color');
// ... etc.
```

setOpacity

Seta a opacidade do elemento.

```
Ext.fly('elId').setOpacity(.5);
Ext.fly('elId').setOpacity(.45, true); // anima
// anima com a duração de .45 segundos
Ext.fly('elId').setOpacity(.45, {duration: .5});
```

clearOpacity

Limpa qualquer configuração de opacidade do elemento.

```
Ext.fly('elId').clearOpacity();
```

Navegação Dom

Freqüentemente nós queremos navegar em torno de uma árvore dom a partir de alguma dada posição na qual nós estamos trabalhando. Ext Core fornece muitos métodos úteis

cross-browser que dão a você um tremendo poder para percorrer para cima, para baixo e tudo em torno da dom. Novamente CSS vem para o resgate quando nós queremos executar uma complexa navegação de forma bastante fácil. O uso de seletores CSS3 tem uma poderosa aplicação neste domínio. Veja a seguinte marcação:

```
<style type="text/css">
    .red {
        color: #F00;
    }
</style>
...
<div id='elId'>
    <ul>
        <li>a-one</li>
        <li>a-two</li>
        <li>a-three</li>
        <li>a-four</li>
    </ul>
    <ul>
        <li>b-one</li>
        <li>b-two</li>
        <li>b-three</li>
    </ul>
</div>
```

Agora digamos que você tenha uma requisição para fazer cada item em uma posição par da lista ter um texto vermelho. Ext Core permite que você resolva isso elegantemente com a seguinte linha.

```
Ext.fly('elId').select('li:nth-child(2n)').addClass('red');
```

Aqui está o resultado:

- a-one
- a-two
- a-three
- a-four

- b-one
- b-two
- b-three

Agora que nós temos visto um pouco do poder contido na API de Navegação DOM da Ext Core, vamos explorar mais:

is

Testa o elemento atual para verificar se ele combina com o seletor.

```
var el = Ext.get('elId');

if (el.is('p.myCls')) {

        // faz alguma coisa

}
```

findParent

Encontra um nó pai que combina com o seletor começando do nó atual.

```
Ext.fly('elId').findParent('div'); // retorna um nó dom

Ext.fly('elId').findParent('div', 4); // olha 4 nós pai acima

Ext.fly('elId').findParent('div',   null,   true);   //   retorna   um
Ext.Element
```

findParentNode

Encontra um nó pai que combina com o seletor começando do nó pai.

```
Ext.fly('elId').findParentNode('div');
```

up

Encontra um nó pai que combina com o seletor, começando do nó pai. Retorna um Ext.Element

```
Ext.fly('elId').up('div');

Ext.fly('elId').up('div', 5); // olha somente 5 pais acima
```

select

Recebe uma consulta e retorna um Ext.CompositeElement ou um array de Ext.Elements. Além disso, se você chamá-la de um elemento este é usado como raiz para começar a consulta, se ele é chamado de Ext.Select o documento inteiro será pesquisado.

```
// CompositeElement dos resultados é retornado

Ext.fly('elId').select('div:nth-child(2)');

// Array de elementos é retornado

Ext.fly('elId').select('div:nth-child(2)', true);

// Todo o documento é pesquisado

Ext.select('div:nth-child(2)');
```

query

Recebe uma consulta e retorna um array de nós dom. Recebe um elemento raiz, opcional, como argumento, se este não estiver presente o default é a raiz do documento.

```
Ext.query('div:nth-child(2)');  // Array de nós dom que combinam com o seletor
```

child

Child seleciona um único filho, em qualquer profundidade abaixo do elemento atual baseado no seletor passado como parâmetro.

```
Ext.fly('elId').child('p.highlight');  // retorna um Ext.Element

Ext.fly('elId').child('p.highlight', true);  // retorna um nó dom
```

down

Seleciona um único filho direto baseado no seletor passado como parâmetro.

```
Ext.fly('elId').down('span');  // retorna um Ext.Element

Ext.fly('elId').down('span', true);  // retorna um nó dom
```

parent

Pega o nó pai deste elemento. Se um seletor é passado como parâmetro ele irá subir na corrente procurando por uma combinação.

```
// agarra o nó pai como Ext.Element
```

```
Ext.fly('elId').parent();

// agarra o nó pai como um nó dom

Ext.fly('elId').parent("", true);

// agarra o primeiro pai que é uma div, retorna Ext.Element

Ext.fly('elId').parent("div");
```

next

Pega o próximo irmão pulando nós de texto. Se um seletor é passado como parâmetro ele irá filtrar por este seletor.

```
// agarra o próximo nó irmão como Ext.Element

Ext.fly('elId').next();

// agarra o próximo nó irmão como um nó dom

Ext.fly('elId').next("", true);

// agarra o próximo nó irmão que é uma div, retorna Ext.Element

Ext.fly('elId').next("div");
```

prev

Pega o irmão anterior pulando nós de texto. Se um seletor é passado como parâmetro ele irá filtrar por este seletor.

```
// agarra o nó irmão anterior como Ext.Element

Ext.fly('elId').prev();

// agarra o nó irmão anterior como um nó dom

Ext.fly('elId').prev("", true);

// agarra o nó irmão anterior que é uma div, retorna Ext.Element

Ext.fly('elId').prev("div");
```

first

Pega o primeiro irmão pulando nós de texto. Se um seletor é passado como parâmetro ele irá filtrar por este seletor.

```
// agarra o primeiro nó irmão como Ext.Element

Ext.fly('elId').first();
```

```
// agarra o primeiro nó irmão como um nó dom

Ext.fly('elId').first("", true);

// agarra o primeiro nó irmão que é uma div, retorna Ext.Element

Ext.fly('elId').first("div");
```

last

Pega o último nó irmão pulando nós de texto. Se um seletor é passado como parâmetro ele irá filtrar por este seletor.

```
// agarra o último nó irmão como Ext.Element

Ext.fly('elId').last();

// agarra o último nó irmão como um nó dom

Ext.fly('elId').last("", true);

// agarra o último nó irmão que é uma div, retorna Ext.Element

Ext.fly('elId').last("div");
```

Manipulação

Elementos

Em uma página dinâmica, uma das mais freqüentes tarefas é criar e remover elementos na DOM. Devido à inconsistência entre os navegadores isso pode provar ser um desafio. Ext Core fornece uma API forte que abstrai essas inconsistências cross-browser e é internamente otimizada para ser eficiente. Nós podemos facilmente adicionar, remover ou substituir nós nessa árvore Dom em qualquer posição que nós queremos relativa a irmãos e pais usando a API de Manipulação Dom da Ext Core.

Vamos dar uma olhada – observe a seguinte marcação:

```
<div id='elId'>

        <p>paragraph one</p>

        <p>paragraph two</p>

        <p>paragraph three</p>

</div>
```

Aqui está um screenshot da marcação:

paragraph one

paragraph two

paragraph three

Vamos criar um novo nó e inseri-lo como o primeiro filho de 'elId':

```
Ext.fly('elId').insertFirst({
        tag: 'p',
        html: 'Hi I am the new first child'
});
```

Agora nós podemos ver nosso nó inserido:

Hi I am the new first child

paragraph one

paragraph two

paragraph three

Muito fácil – vamos explorar algo mais da poderosa API de manipulação oferecida pela Ext Core:

appendChild

Anexa o(s) elemento(s) passado(s) como parâmetro a este elemento.

```
var el = Ext.get('elId1');
// anexa o nó dom com este id.
Ext.fly('elId').appendChild('elId2');
// anexa um Ext.Element
Ext.fly('elId').appendChild(el);
// anexa o resultado de um array de seletores
Ext.fly('elId').appendChild(['elId2','elId3']);
// anexa um nó dom
Ext.fly('elId').appendChild(el.dom);
```

```
// anexa um CompositeElement de todas as divs
Ext.fly('elId').appendChild(Ext.select('div'));
```

appendTo

Anexa este elemento ao elemento passado como parâmetro.

```
var el = Ext.get('elId1');

Ext.fly('elId').appendTo('elId2');  // anexa o nó dom com este id.
Ext.fly('elId').appendTo(el);  // anexa el ao Ext.Element
```

insertBefore

Insere este elemento antes do elemento passado como parâmetro.

```
var el = Ext.get('elId1');

// insere antes do nó dom.
Ext.fly('elId').insertBefore('elId2');
// insere el antes do Ext.Element
Ext.fly('elId').insertBefore(el);
```

insertAfter

Insere este elemento depois do elemento passado como parâmetro.

```
var el = Ext.get('elId1');

// insere depois do nó dom com este id.
Ext.fly('elId').insertAfter('elId2');
// insere el depois do Ext.Element
Ext.fly('elId').insertAfter(el);
```

insertFirst

Recebe um elemento, id ou uma configuração DomHelper e insere o nó existente ou cria um novo elemento a partir da configuração DomHelper como o primeiro filho deste elemento.

```
var el = Ext.get('elId1');


// insere o nó dom com este id como o seu primeiro filho.

Ext.fly('elId').insertFirst('elId2');

// insere o Ext.Element el como o primeiro filho

Ext.fly('elId').insertFirst(el);


// Cria um novo nó com a configuração DomHelper e insere o resultado
//como primeiro filho.

Ext.fly('elId').insertFirst({

        tag: 'p',

        cls: 'myCls',

        html: 'Hi I am the new first child'

});
```

replace

Substitui o elemento passado como parâmetro por este elemento.

```
var el = Ext.get('elId1');


 // substitui o elemento com o id 'elId2' pelo 'elId'.

Ext.fly('elId').replace('elId2');

// substitui o Ext.Element el pelo 'elId'

Ext.fly('elId').replace(el);
```

replaceWith

Recebe um elemento, id ou configuração DomHelper e substitui este elemento pelo nó existente ou novo elemento criado a partir da configuração DomHelper.

```
var el = Ext.get('elId1');
```

```
Ext.fly('elId').replaceWith('elId2');      //   substitui   'elId'   por
'elId2'.

Ext.fly('elId').replaceWith(el);  // substitui 'elId' por 'elId1'


// Cria um novo nó com a configuração DomHelper

// e substitui 'elId' por este nó.

Ext.fly('elId').replaceWith({

        tag: 'p',

        cls: 'myCls',

        html: 'Hi I have replaced elId'

});
```

Configurações DomHelper

Nos exemplos acima, talvez você tenha obvservado o seguinte:

```
.insertFirst({

        tag: 'p',

        html: 'Hi I am the new first child'

});
```

Você pode ter se maravilhado com o que o primeiro argumento de insertFirst é. Esta é uma configuração DomHelper que representa a marcação a ser criada. Configurações DomHelper suportam muitas propriedades para especificar nós filhos, tais como fragmentos HTML e atributos DomNode (classes css, url, src, id, etc). Aqui estão algumas das APIs disponíveis em Ext.Element que te permitem interagir diretamente com o Ext.DomHelper:

createChild

Cria a configuração DomHelper passada como parâmetro e a anexa a este elemento ou opcionalmente insere antes do elemento filho passado como parâmetro.

```
var el = Ext.get('elId');

var dhConfig = {

        tag: 'p',

        cls: 'myCls',

        html: 'Hi I have replaced elId'
```

```
};


// Cria um novo nó e o anexa a 'elId'.
el.createChild(dhConfig);
// Cria um novo nó e o insere antes do primeiro filho de el.
el.createChild(dhConfig, el.first());
```

wrap

Embrulha este elemento com o novo elemento criado

```
Ext.fly('elId').wrap(); // embrulha elId com uma div


// Embrulha elId com o novo elemento criado
Ext.fly('elId').wrap({
        tag: 'p',
        cls: 'myCls',
        html: 'Hi I have replaced elId'
});
```

Fragmentos Html

Fragmentos Html são nada mais do que eles soam, fragmentos de marcação HTML. Ext Core fornece a habilidade para modificar a dom com fragmentos HTML, permitindo que você se concentre apenas nos fragmentos de marcação com os quais você pretende modificar o dom, liberando você da implementação do navegador e das preocupações com performance.Ext Core faz de tudo pra você – tudo o que você precisa fazer é fornecer a marcação. Por exemplo, dada a seguinte marcação:

```
<div id='elId'>
    <li>one</li>
    <li>two</li>
    <li>three</li>
    <li>four</li>
</div>
```

O que você acha que o seguinte código da Ext Core irá fazer?

```
Ext.fly('elId').insertHtml('beforeBegin', '<p>Hi</p>')
```

Vamos dar uma olhada na marcação seguinte:

```
<p>Hi</p>
<div id='elId'>
    <li>one</li>
    <li>two</li>
    <li>three</li>
    <li>four</li>
</div>
```

Surpreso? Isto é porque nós somos capazes de especificar uma posição relativa onde nós queremos que a marcação seja aplicada. Nós especificamos com 'beforeBegin'. E sobre esta:

```
Ext.fly('elId').insertHtml('afterBegin', '<p>Hi</p>')
```

Vamos dar uma olhada:

```
<div id='elId'>
    <p>Hi</p>
    <li>one</li>
    <li>two</li>
    <li>three</li>
    <li>four</li>
</div>
```

Agora nós iremos usar 'beforeEnd'.

```
Ext.fly('elId').insertHtml('beforeEnd', '<p>Hi</p>')
```

Vamos dar uma olhada:

```
<div id='elId'>
    <li>one</li>
```

```
    <li>two</li>
    <li>three</li>
    <li>four</li>
    <p>Hi</p>
</div>
```

E finalmente vamos dar uma olhada em 'afterEnd'.

```
Ext.fly('elId').insertHtml('beforeEnd', '<p>Hi</p>')
```

Vamos dar uma olhada:

```
<div id='elId'>
    <li>one</li>
    <li>two</li>
    <li>three</li>
    <li>four</li>
</div>
<p>Hi</p>
```

A API seguinte é destinada para você trabalhar com fragmentos html.

insertHtml

Insere um fragmento HTML dentro deste elemento. Você precisa especificar onde inserir o elemento (beforeBegin, beforeEnd, afterBegin, afterEnd) e pode opcionalmente, se você quiser, retornar um Ext.Element do nó dom.

```
Ext.fly('elId').insertHtml(
        'beforeBegin',
        '<p><a href="anotherpage.html'>click me</a></p>'
); // retorna um nó dom
Ext.fly('elId').insertHtml(
        'beforeBegin',
        '<p><a href="anotherpage.html'>click me</a></p>',
        true
```

```
); // retorna um Ext.Element
```

remove

Remove este elemento da DOM e o deleta da cache.

```
Ext.fly('elId').remove(); // elId é removido da dom e da cache
```

removeNode

Remove um nó DOM do documento. O corpo do nó será ignorado se for passado como parâmetro.

```
Ext.removeNode(node); // nó (HTMLElement) é removido da dom
```

Ajax

Ext Core inclui uma poderosa API Ajax. Ajax será coberto em maiores detalhes em seções futuras, mas aqui está uma breve visão panorâmica da presente API na Ext Core:

load

Acesso direto ao método Ext.Updater.update do Updater. O método recebe o mesmo parâmetro do objeto assim como Ext.Updater.update

```
Ext.fly('elId').load({url: 'serverSide.php'})
```

getUpdater

Pega o Ext.Updater deste elemento

```
var updr = Ext.fly('elId').getUpdater();

updr.update({
        url: 'http://myserver.com/index.php',
        params: {
            param1: "foo",
            param2: "bar"
        }
});
```

Manipulação de eventos

A manipulação de eventos coloca uma das mais problemáticas diferenças entre navegadores. Ext Core abstrai as diferenças que você deve encontrar para alcançar a manipulação de eventos cross-browser. Se você fosse utilizar cada mecanismo nativo de manipulação de eventos do navegador você iria usar métodos diferentes para registrar e desregistrar eventos em adição à miríade de outras inconsistências frustrantes.

Ext Core fornece um rico modelo de eventos que te liberta dessas diferenças pelo uso de uma interface simples, limpa e consistente. Assim como Ext.Element envolve o nó Dom nativo, Ext Core irá também envolver o objeto-evento nativo do navegador com uma instância de Ext.EventObject. Ext.EventObject normaliza as diferenças cross-browser, tais como as a manipulação de um botão clicado, teclas pressionadas, mecanismos para parar a propagação de um evento juntamente com um método para prevenir as ações default a partir de um dado lugar.

Para amarrar um manipulador de eventos a um elemento em uma página nós iremos usar o método **on** de Ext.Element. O método **on** é um atalho para **addListener**. O primeiro argumento especifica a qual evento se inscrever e o segundo argumento especifica a função a ser executada quando o evento ocorrer.

```
Ext.fly('myEl').on('click', function(e, t) {

        // executa uma ação no on click de myEl

        // e é um Ext.EventObject descrevendo o que ocorreu

        // t é o HTMLElement albo

});
```

Ext Core normaliza os argumentos de todos os eventos Dom. O manipulador de eventos que é amarrado ao evento irá sempre receber um objeto de eventos normalizado (Ext.EventObject) e o HTMLElement alvo.

Vamos dar uma olhada na API fornecida para a Manipulação de Eventos:

addListener/on

Anexa um manipulador de evento a este elemento. A versão atalho on é equivalemente e é preferível para um código conciso.

```
var el = Ext.get('elId');

el.on('click', function(e,t) {

        // e é um objeto de evento normalizado (Ext.EventObject)

        // t o alvo que foi clicado, este é um Ext.Element.

        // esse também aponta para t.

});
```

removeListener/un

Remove um manipulador de eventos deste elemento. A versão atalho une equivalente.

```
var el = Ext.get('elId');

el.un('click', this.handlerFn);

// ou

el.removeListener('click', this.handlerFn);
```

Ext.EventObject

EventObject expõe um modelo de evento cross-browser consistente mimetizando o método padrão W3C se necessário.

```
// e não é um objeto de evento padrão, ele é um Ext.EventObject

function handleClick(e){

    e.preventDefault();

    var target = e.getTarget();

    ...

}

var myDiv = Ext.get('myDiv');

myDiv.on("click", handleClick);

//ou

Ext.EventManager.on('myDiv', 'click', handleClick);

Ext.EventManager.addListener('myDiv', 'click', handleClick);
```

Manipulação de eventos avançada

Ext Core tem muitas configurações convenientes para técnicas de manipulação de eventos mais avançadas tais como delegação de eventos, buferização e atrasos.

delegação

Delegação de eventos é uma técnica que é usada para reduzir o consumo de memória e prevenir a exposição a vazamento de memória. A idéia básica é esta:

Ao invés de registrar um manipulador de eventos para cada elemento em um grupo de elementos, registre-o uma vez em um container de elementos aproveitando a vantagem da fase de bolha no ciclo de vida do evento e centralizando a manipulação de eventos no nível do container.

Isso não significa que você quer registrar um manipulador global no elemento corpo, como qualquer ação em uma página que acionou um evento deve subir ao manipulador e provavelmente tem o efeito inverso, realmente degradando a performance. Essa técnica é

extremamente útil em situações como drop-down, um calendário, etc. Em qualquer parte que você tiver um grupo de elementos que podem ser diretamente ou quase diretamente contido por um elemento container. Vamos dar uma olhada:

Tome a seguinte marcação:

```
<ul id='actions'>
    <li id='btn-edit'></li>
    <li id='btn-delete'></li>
    <li id='btn-cancel'></li>
</ul>
```

Em vez de registrar um manipulador diretamente sobre cada item da lista, como se segue:

```
Ext.fly('btn-edit').on('click, function(e,t) {
        // manipula o evento
});
Ext.fly('btn-delete').on('click, function(e,t) {
        // manipula o evento
});
Ext.fly('btn-cancel').on('click, function(e,t) {
        // manipula o evento
});
```

Para usar a técnica de delegação de eventos em vez de registrar um manipulador em todo o container juntamente com a lógica de acompanhamento:

```
Ext.fly('actions').on('click, function(e,t) {
        switch(t.id) {
                case 'btn-edit':
                // manipula o evento
                break;
                case 'btn-delete':
                // manipula o evento
```

```
                break;
                case 'btn-cancel':
                // manipula o evento
                break;
        }
});
```

Pelo fato dos eventos subirem como bolhas para cima na hierarquia dom, eles irão atingir o manipulador que nós registramos na div 'actions'. Nós então usamos um simples switch para executar o código requerido. Esta abordagem funciona bem porque nós podemos adicionar muitos diferentes elementos filhos e ainda manter um manipulador de eventos simples.

delegate

Esta é uma opção de configuração que você pode passar junto quando está registrando um manipulador para um evento para ajudar no evento de delegação. Pela definição desta opção de configuração por um simples seletor, Ext Core irá filtrar o alvo ou procurar por um descendente deste alvo.

```
el.on('click', function(e,t) {
    // manipula click
}, this, {
    // irá filtrar o alvo a ser um descendente com a classe 'clickable'
    delegate: '.clickable'
});
```

hover

Ext Core tem uma solução cross-browser para criar um efeito hover. O método hover irá certificar-se de que uma função é executada quando o mouse entrar e um elemento e quando o mouse deixar um elemento. mouseenter e mouseleave são eventos nativos do Internet Explorer que irão filtrar os eventos mouseout e mouseover quando você estiver trabalhando com elementos filhos. ExtCore implementa esses eventos para navegadores condizentes com W3C, dessa forma você pode usá-lo em qualquer navegador.

```
// manipula quando o mouse entra no elemento
function enter(e,t){
    t.toggleClass('red');
```

```
}

// manipula quando o mouse deixa o elemento

function leave(e,t){

   t.toggleClass('red');

}

// increve-se no hover

el.hover(over, out);
```

removeAllListeners

Remove todos os listeners anteriormente adicionados deste elemento

```
el.removeAllListeners();
```

single

Está é uma opção de configuração que você pode passar junto quando está registrando um manipulador para o evento. Quando setada para true, o manipulador irá se acionar somente uma vez e então se remover automaticamente.

```
el.on('click', function(e,t) {

    // manipula click

}, this, {

    single: true // irá remover o evento depois do seu primeiro acionamento.

});
```

buffer

Está é uma opção de configuração que você pode passar junto quando está registrando um manipulador para o evento. Ela faz com que o manipulador seja agendado para a execução em um Ext.util.DelayedTask esperando por um número especificado de milisegundos. Se o evento acionar novamente dentro deste tempo, o manipulador original não será invocado, mas um novo manipulador é agendado no seu lugar.

```
el.on('click', function(e,t) {

    // manipula click

}, this, {
```

```
    buffer: 1000 // irá esperar e armazenar o acionamento deste evento
//para depois, ele inicialmente aciona com 1000 milisegundos (ou 1
segundo).

});
```

delay

Está é uma opção de configuração que você pode passar junto quando está registrando um manipulador para o evento. O número de milisegundos para atrasar a invocação do manipulador depois de acionado.

```
el.on('click', function(e,t) {

    // manipula click

}, this, {

    // irá atrasar o acionamento do evento após sua primeira execução

    // 1000 milisegundos (ou 1 segundos).

    delay: 1000

    });
```

target

Está é uma opção de configuração que você pode passar junto quando está registrando um manipulador para o evento. Se você tem um alvo específico que você quer manipular, você pode atribuir uma referência dele nesta opção de configuração e Ext Core irá certificar-se de que seu manipulador receba somente chamadas do manipulador quando ele for disparado deste nó na fase de bolha.

```
el.on('click', function(e,t) {

    // manipula click

}, this, {

    // somente manipula o evento quando ele foi originado da primeira
‘div’.

    target: el.up('div')

});
```

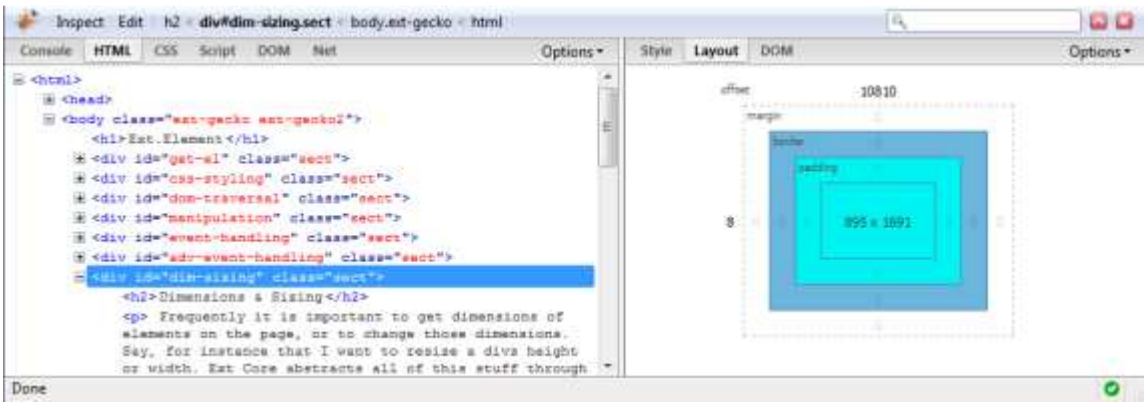
Dimensões e Dimensionamento

Freqüentemente nós necessitamos recuperar ou modificar as dimensões de algum elemento particular em uma página. Mais uma vez, Ext Core abstrai o dimensionamento através de

uma API limpa. A maioria dos métodos setters aceitam configurações de animação ou uma sinalização booleana para animar a ação. Vamos dar uma olhada:

```
//  muda a altura para 200px e anima com a configuração default

Ext.fly('elId').setHeight(200, true);


// muda a altura para 150px e anima com configurações customizadas

Ext.fly('elId').setHeight(150, {

    duration : .5, // a animação irá ter uma duração de .5 segundo

    // ira mudar o conteúdo para finalizado

    callback: function(){ this.update("finished"); }

});
```

Sobre o firebug, inspect e element (clique direito e então clique em 'InspectElement') e vamos dar uma olhada no painel da direita e clicar em "layout". Você deve ver algo como isso:



Esta informação é extremamente útil uma vez que ela fornece todas as dimensões do elemento inspecionado. Observe que o elemento tem uma largura de 895px, uma altura de 1691px, espaçamento de 0px em todo seu redor, borda de 0px em todo seu redor e 0px de margem. Toda esta informação está disponível para você através da API de Dimensionamento do Ext.Element da Ext Core!

```
var dimSz = Ext.get('dim-sizing');

var padding = dimSz.getPadding('lrtb'); // tem valor de 0px

var border = dimSz.getBorderWidth('lrtb'); // tem valor de 0px

var height = dimSz.getHeight(); // tem valor de 1691px

var width = dimSz.getWidth(); // tem valor de 895px
```

Pegue este código e execute-o no firebug para ver por si mesmo. De fato iremos um passo a frente e tentaremos usar os setters para modificar a altura e largura e ver o que acontece com a caixa no painel de layout do firebug. (**Observe:** se sua altura e largura são direfentes da imagem, isso é por causa das dimensões do seu navegador. Se você inspecionar o elemento em seu navegador real a saída irá combinar.)

Vamos explorar o resto da API:

getHeight

Retorna a altura do elemento

```
var ht = Ext.fly('elId').getHeight();
```

getWidth

Retorna a largura do elemento

```
var wd = Ext.fly('elId').getWidth();
```

setHeight

Seta a altura do elemento

```
Ext.fly('elId').setHeight();
```

setWidth

Seta a largura do elemento

```
Ext.fly('elId').setWidth();
```

getBorderWidth

Pega a largura das bordas para os lados especificados. Lados podem ser t, l, r, b ou qualquer combinação desses para adicionar múltiplos valores. Por Exemplo, passando lr você irá pegar a largura da borda (l)eft + a largura da borda (r)ight.

```
var bdr_wd = Ext.fly('elId').getBorderWidth('lr');
```

getPadding

Pega a largura do espeçamento dos lados especificados. Lados podem ser t, l, r, b ou qualquer combinação desses para adicionar múltiplos valores. Por Exemplo, passando lr você irá pegar a largura do espaçamento (l)eft + a largura do espaçamento (r)right.

```
var padding = Ext.fly('elId').getPadding('lr');
```

clip

Armazena a configuração de overflow corrente e prende o overflow no elemento – use unclip para remover

```
Ext.fly('elId').clip();
```

unclip

Retorna o overflow preso para o overflow orginal antes da função clip() ser chamada

```
Ext.fly('elId').unclip();
```

isBorderBox

True se o browser detectado é o internet Explorer rodando em um modo não estrito.

```
if (Ext.isBorderBox) {

    // faz alguma coisa

}
```

Posicionamento

Através da API de posicionamento da Ext Core é possível recuperar e setar todos os aspectos do posicionamento de um elemento de todos os navegadores. Similar à API de dimensionamento, a maioria dos setters suportam animação, quer através de um booleano true (para o default) ou pela passagem de uma configuração de objeto literal como segundo argumento. Vamos ver um exemplo de como isso é:

```
// modifica a coordenada x para 75px e anima com uma configuração customizada

Ext.fly('elId').setX(75, {

    duration : .5, // animaçaõ irá ter uma duração de .5 segundos
// irá mudar o conteúdo para “finished”

    callback: function(){ this.update("finished"); }

});
```

getX

Pega a posição x atual do elemento baseado nas coordenadas da página. O elemento precisa ser parte de uma árvore DOM para ter coordenadas da página (display:none ou elementos não anexados retornam false)

```
var elX = Ext.fly('elId').getX()
```

getY

Pega a posição y do elemento baseado nas coordenadas da página. O elemento precisa ser parte de uma árvore DOM para ter coordenadas da página (display: none ou elementos não anexados retornam false)

```
var elY = Ext.fly('elId').getY()
```

getXY

Pega a posição corrente do elemento baseado nas coordenadas da página. O elemento precisa ser parte de uma árvore DOM para ter coordenadas da página (display: none ou elementos não anexados retornam false).

```
var elXY = Ext.fly('elId').getXY() // elXY é um array
```

setX

Seta a posição X do elemento baseado nas coordenadas da página. Elementos precisam ser parte de uma árvore DOM para ter coordenadas da página (display: none ou elementos não anexados retornam false).

```
Ext.fly('elId').setX(10)
```

setY

Seta a posição Y de um elemento baseado nas coordenadas da página. O Elemento precisa ser parte de uma árvore dom para ter coordenadas da página (display: none ou elementos não anexados retornam false).

```
Ext.fly('elId').setY(10)
```

setXY

Seta a posição do elemento na coordenada da página indiferentemente de como o elemento está posicionado. O elemento precisa ser parte de uma árvore DOM para ter coordenadas da página (display: none ou elementos não anexados retornam false).

```
Ext.fly('elId').setXY([20,10])
```

getOffsetsTo

Retorna a distância deste elemento a partir do elemento passado. Ambos os elementos precisam ser parte de uma árvore DOM e não terem display:none definido para ter coordenadas da página.

```
var elOffsets = Ext.fly('elId').getOffsetsTo(anotherEl);
```

getLeft

Pega a coordenda X esquerda do elemento

```
var elLeft = Ext.fly('elId').getLeft();
```

getRight

Pega coordenada X direita do elemento (posição X do elemento + largura do elemento)

```
var elRight = Ext.fly('elId').getRight();
```

getTop

Pega a coordenada Y do topo do elemento

```
var elTop = Ext.fly('elId').getTop();
```

getBottom

Pega a coordenada Y da base do elemento (posição Y do elemento+ altura do elemento)

```
var elBottom = Ext.fly('elId').getBottom();
```

setLeft

Seta a posição esquerda do elemento diretamente usando estilos CSS (em vez de setX).

```
Ext.fly('elId').setLeft(25)
```

setRight

Seta o estilo CSS direita do elemento.

```
Ext.fly('elId').setRight(15)
```

setTop

Seta a posição topo do elemento usando diretamente estilos CSS (em vez de suar setY)

```
Ext.fly('elId').setTop(12)
```

setBottom

Seta o estilo CSS da base do elemento.

```
Ext.fly('elId').setBottom(15)
```

setLocation

Seta a posição do elemento nas coordenadas da página, indiferentemente de como o elemento está posicionado. O elemento precisa ser parte de uma árvore DOM para ter coordenadas da página (display: none ou elementos não anexados retornam false)

```
Ext.fly('elId').setLocation(15,32)
```

moveTo

Seta a posição do elemento nas coordenadas da página, indiferentemente de como o elemento está posicionado. O elemento precisa ser parte de uma árvore DOM para ter coordenadas da página (display: none ou elementos não anexados retornam false)

```
Ext.fly('elId').moveTo(12,17)
```

position

Inicializa o posicionamento neste elemento. Se uma posição desejada não é passada, ele irá fazer o posicionamento relativo do elemento se ele já não estiver posicionado.

```
Ext.fly('elId').position("relative")
```

clearPositioning

Limpa o posicionamento de volta para o default de quando o elemento foi carregado

```
Ext.fly('elId').clearPositioning()

Ext.fly('elId').clearPositioning("top")
```

getPositioning

Recupera um objeto com todas as propriedades de posicionamento CSS. Útil junto com o setPositioning() para pegar um snapshot antes de executar uma atualização e então restaurar o elemento.

```
var pos = Ext.fly('elId').getPositioning()
```

setPositioning

Seta o posicionamento com um objeto retornado pelo getPositioning ().

```
Ext.fly('elId').setPositioning({
    left: 'static',
    right: 'auto'
})
```

translatePoints

Traduz as coordendas da página passada em valores left/top para este elemento

```
// {left:translX, top: translY}
var points = Ext.fly('elId').translatePoints(15,18);
```

Animações

Ext Core tem disponíveis plugins de animação que dão a você um conjunto robusto de animações pré-configuradas que são aplicadas como métodos para Ext.Element assim você pode fazer coisas legais como isso:

```
Ext.fly('slideEl').slideOut('r');
```

Copie o código acima, execute-o no FireBug e veja o que acontece. Você irá ver abaixo que Ext já tem um conjunto completo de animações embutidas para você. Cada animação recebe um objeto literal de configuração tornando-o fácil e muito customizável, se você precisar mais do que o comportamento default. Talvez você queira adicionar sua própria função de retorno para acionar sobre a conclusão da animação:

```
Ext.fly('slideEl').slideOut('r', {
        callback : function(){
                alert('Finished sliding the element out');
        }
});
```

Assim você pode ver que há algumas animações realmente poderosas aqui.

Animações suportam 8 maneiras de ancoragem, assim você pode escolher a partir de 8 diferentes pontos de ancoramento para começar e terminar sua animação.

Descrição dos valores

Descrição dos valores

tl	O canto superior esquerdo
t	O centro do lado superior
tr	O canto superior direito
l	O centro do lado esquerdo
r	O centro do lado direito
bl	O canto inferior esquerdo
b	O centro do lado inferior
br	O canto inferior direito

Vamos explorar melhor a API:

slideIn/slideOut

Escorrega o elemento dentro ou fora da visão. Um ponto de ancoramento é opcionalmente passado para o ponto de origem para o efeito de escorregamento. Esta função automaticamente manipula o invólucro do elemento com um container de tamanho fixo se necessário. Veja a visão panorâmica da classe Fx para ver as opções de ponto de ancoramento válidas. Uso:

```
// default: escorrega o elemento a partir do topo
el.slideIn();
// default: escorrega o elemento para fora a partir da sua base
el.slideOut();

// opções de configuração comuns mostradas com valores default
el.slideIn('t', {
    easing: 'easeOut',
    duration: .5
});
el.slideOut('t', {
    easing: 'easeOut',
    duration: .5,
```

```
    remove: false,
    useDisplay: false
});
```

puff

Esfumaça o elemento para fora enquanto o expande devagar em todas as direções. Quando o efeito é concluído, o elemento será escondido (visibility = 'hidden'), mas o bloco do elemento irá receber um espaço em branco no documento. O elemento precisa ser removido da DOM usando a opção de configuração 'remove' se desejado. Uso:

```
// default
el.puff();

//opções de configuração comuns mostradas com valores default
el.puff({
    easing: 'easeOut',
    duration: .5,
    remove: false,
    useDisplay: false
});
```

switchOff

Pisca o elemento como se ele tivesse sido clicado e então o esconde em seu centro (similar ao desligamento de uma televisão). Quando o efeito é concluído, o elemento será escondido (visibility = 'hidden') mas o bloco do elemento irá receber um espaço em branco no documento. O elemento precisa ser removido da DOM usando a opção de configuração 'remove' se desejado. Uso:

```
// default
el.switchOff();

// todas as opções de configuração mostradas com valores default
el.switchOff({
    easing: 'easeIn',
    duration: .3,
```

```
    remove: false,
    useDisplay: false
});
```

highlight

Destaca o elemento pela definição de uma cor (aplica a cor ao background-color pelo default, mas pode ser modificada usando a opção de configuração 'attr') e então esfumaça o elemento de volta à sua cor original. Se nenhuma cor original estiver disponível, você deve fornecer uma opção de configuração de cor final "endColor"que será limpa depois da animação. Uso:

```
// default: destaca o background para amarelo
el.highlight();

// opções de configuração comuns mostradas com valores default
el.highlight("ffff9c", {
    //pode ser qualquer propriedade CSS válida (atributo) que suporta um valor de cor
    attr: "background-color",
    endColor: (current color) or "ffffff",
    easing: 'easeIn',
    duration: 1
});
```

frame

Mostra uma ondulação de explosão, atenuando as bordas para chamar a atenção para um elemento. Uso:

```
// default: uma simples ondulação de luz azul
el.frame();

// opções de configuraçãoo comuns mostradas com valores default
el.frame("C3DAF9", 1, {
    duration: 1 //duração de cada ondução individual.
```

```
    // Observe: Flexibilização não é configurável e será ignorada se incluída
});
```

pause

Cria uma pausa antes de qualquer efeito na fila começar. Se não há efeitos na fila depois da pausa, ele não terá efeito. Uso:

```
el.pause(1);
```

fadeIn/fadeOut

Desbota um elemento (de transparente para opaco). A opacidade final pode ser ser especificada usando a opção de configuração "endOpacity". Ou inverte o desbotamento (de opaco para transparente). A opacidade final pode ser especificada usando a opção de configuração "endOpacity". Observe que o IE pode requerer useDisplay: true para re-exibir corretamente. Uso:

```
// default: fade in de opacidade 0 para 100%
el.fadeIn();
el.fadeOut();


// opções de configuração comuns mostradas com valores default
el.fadeIn({
    endOpacity: 1, //pode ser qualquer valor entre 0 e 1 (ex: .5)
    easing: 'easeOut',
    duration: .5
});
el.fadeOut({
    endOpacity: 0, // pode ser qualquer valor entre 0 e 1 (ex: .5)
    easing: 'easeOut',
    duration: .5,
    remove: false,
    useDisplay: false
});
```

scale

Anima a transição de dimensões de um elemento a partir de uma altura/largura inicial para uma altura/largura final. Uso:

```
// modifica a altura e largura para 100x100 pixels

el.scale(100, 100);


// opções de configuração comuns mostradas com valores default. A altura e largura serão

// default para os valores existentes do elemento se forem passadas como null.

el.scale(

    [element's width],

    [element's height], {

            easing: 'easeOut',

            duration: .35

        }

);
```

shift

Anima a transição de uma combinação das dimensões, posições xy e/ou opacidade. Qualquer dessas propriedades não especificadas no objeto de configuração não serão modificadas. Este efeito requer que ao menos uma nova configuração de dimensão, posição ou opacidade sejam passadas no objeto de configuração para que a função tenha qualquer efeito. Uso

```
// escorrega o elemento horizontalmente para a posição x 200 enquanto
// muda a altura e opacidade

el.shift({ x: 200, height: 50, opacity: .8 });


// opções de configuração comuns mostradas com valores default.

el.shift({

    width: [element's width],

    height: [element's height],

    x: [element's x position],
```

```
    y: [element's y position],
    opacity: [element's opacity],
    easing: 'easeOut',
    duration: .35
});
```

ghost

Escorrega o elemento enquanto o desbota para fora da visão. Um ponto de ancoramento pode ser opcionalmente passado para definir o ponto de término do efeito. Uso:

```
// default: escorrega o elemento para baixo enquanto o desbota
el.ghost();

// opções de configuração comuns mostradas com valores default
el.ghost('b', {
    easing: 'easeOut',
    duration: .5,
    remove: false,
    useDisplay: false
});
```

Animações complexas

Nós também podemos usar o sistema de animações Ext Core para construir nossas próprias complexas e customizadas animações. Copie o seguinte código para o FireBug e veja o que ele faz:

```
var el = Ext.get('complexEl')
el.animate({
        borderWidth: {to: 3, from: 0},
        opacity: {to: .3, from: 1},
        height: {to: 50, from: el.getHeight()},
        width: {to: 300, from: el.getWidth()}
});
```

Miscelânea

A seguir estão métodos úteis de Ext.Element que não se ajustam às seções anteriores, mas merecem menção.

focus

Tenta colocar o foco no elemento. Quaisquer exceções são capturadas e ignoradas.

```
el.focus();
```

blur

Tenta remover o foco de um elemento. Quaisquer exceções são capturadas e ignoradas

```
el.blur();
```

getValue

Retorna o valor do atributo "value"

```
el.getValue();

el.getValue(true); // converte o valor para número
```

isBorderBox

Verdadeiro se for detectado que o navegador é o Internet Explorer rodando em um modo não estrito

```
if (Ext.isBorderBox) { }
```

getAttributeNS

Retorna o valor de um atributo nomeado a partir de um nó de elemento sublinhado na DOM.

```
el.getAttributeNS("","name");
```

CompositeElement

O que é um composto?

" O Composite (Padrão de projeto) permite que um grupo de objetos seja tratado da mesma maneira que uma instância simples de um objeto.” (Wikipedia) Ext.CompositeElement permite que você trabalhe com uma coleção de zero a muitos elementos como uma entidade única. ComositeElement usa a mesma interface de Ext.Element simplificando seu trabalho como desenvolvedor eliminando a necessidade de verificações comuns em tempo de execução quando lida com uma coleção. CompositElements são tipicamente recuperados usando o método estátivo Ext.select. Ext.select usa DomQuery para procurar em um documento inteiro por qualquer elemento que combine com um seletor particular.

Por exemplo, dada a seguinte marcação:

```
<html>

   <body>

      <div id="first" class="title">Sample A</div>

      <div id="second" class="doesNotMatch">Lorem Ipsum</div>

      <div id="third" class="title secondCSSCls">Some additional
content</div>

   </body>

</html>
```

Nós poderíamos usar o seletor ‘title” para encontrar na página inteira e recuperar um CompositeElement que contém referências à div ‘first’ e ‘third’.

```
var els = Ext.select('.title');
```

Observe: o elemento ‘third’ também tem uma classe CSS adicional de secondCSSCks. Elementos HTML podem ter múltiplas classes css pela separação de um espaço em branco. Este seletor não requer que somente ‘title’ esteja aplicado e desta forma ambos ‘first’ e ‘third’ serão retornados.

Depois de obtido um CompositeElement nós podemos agora trabalhar com esta coleção de elementos como um único elemento. Por exemplo, nós podemos então adicionar uma classe CSS de erro à coleção.

```
var els = Ext.select('.title');

els.addClass('error');
```

Quando você está ciente da localização dos elementos você pode querer recuperar relativo a outro elemento na página você pode primeiro obter uma referência para este elemento e então procurar por ele. Isso irá fazer a performance da sua pesquisa mais rápida porque ele é procurado em um pequeno conjunto do documento inteiro. Vamos ver o que acontece quando nós adicionamos uma div adicional com um id de ‘accordion’ envolvendo todos os nossos elementos ‘first’, ‘second’ e ‘third’.

```
<html>
   <body>
      <div id="accordion">
         <div id="first" class="title">Sample A</div>
         <div id="second" class="doesNotMatch">Lorem Ipsum</div>
         <div id="third" class="title secondCSSCls">
            Some additional content
         </div>
      </div>
   </body>
</html>
```

Desde que nós sabemos que esses elementos estarão dentro de uma div com o id 'accordion' nós podemos então definir o nosso escopo de pesquisa para somente o elemento accordion.

```
var accordion = Ext.get('accordion');
accordion.select('title');
```

Toda vez que você está ciente da localização dos elementos dentro da sua marcação você deve certificar-se de que seu escopo otimiza a performance.

Outros métodos úteis de CompositeElement são mostrados no código exemplo:

```
var accordion = Ext.get('accordion');
accordion.select('title');
// firstItem agora tem um Ext.Element apontando para a primeira div
var firstItem = accordion.item(0);
 // alerta 1 ou a segunda posição
alert(accordion.indexOf('third'));
// alerta 2
alert(accordion.getCount());
// Remove o elemento de uma coleção e da DOM
accordion.removeElement('one', true);
```

Observe: Métodos que tem sido removidos de CompositeElement que usuários de ExtJS podem estar familiarizados com os mesmos: each, first, last, fill, contains, filter.

Implementação Ajax

Ajax definido

Asynchronous JavaScript and XML (definição de AJAX in 2005), é um grupo de técnicas de desenvolvimento we b inter-relacionadas usadas para criar aplicações web interativas ou aplicações ricas para a internet. Com Ajax, aplicações web podem recuperar dados de um servidor assincronamente em background sem interferir na exibição e comportamento de uma página existente. Dados são recuperados usando o objeto XHR (XMLHttpRequest). Devido à complexidade das distinções cross-browser entre as implementações XHR, frameworks Ajax tem surgido para abstrari essas diferenças em um conjunto de construções de programação reutilizáveis. Na Ext, a execução dessas funções é feita dentro do objeto singleton Ext.Ajax.

Ext.Ajax

Ext.Ajax estende a classe Ext.data.Connection em um objeto singleton e fornece uma área centralizada para fazer requisições Ajax com o máximo de flexibilidade. Através do uso deste singleton, todas as requisições Ajax podem ser roteadas através de uma classe e fornecer acesso a todas as funções, eventos e parâmetros.

Eventos de Ext.Ajax

Ext.Ajax expõem eventos Ajax globais que podem ser manipulados por cada requisição.

beforerequest (conn, opts)

É acionado antes de uma requisição Ajax ser enviada.

requestcomplete (conn, response, opts)

É acionado depois de um requisição Ajax ser concluída com sucesso.

requestexception (conn, response, opts)

É acionado se um status de erro HTTP foi retornado pelo servidor.

```
// Exemplo: mostra um spinner durante todas as requisições Ajax

Ext.Ajax.on('beforerequest', this.showSpinner, this);

Ext.Ajax.on('requestcomplete', this.hideSpinner, this);

Ext.Ajax.on('requestexception', this.hideSpinner, this);
```

Propriedades de Ext.Ajax

Uma vez que Ext.Ajax é um objeto singleton, você pode definir propriedades para ele uma vez e sobrescrevê-lo no nível da função de requisição, se necessário. Propriedades comuns que você pode definir são:

method: O método HTTP default usado para as requisições. Observe que ele é case-sensitive e deve ser tudo em maiúsculo (o default é undefined; se não for definido mas parâmetros estiverem presentes irá usar "POST", senão "GET".)

extraParams: Um objeto contendo propriedades que serão usadas como parâmetros extra para cada requisição feita por este objeto (o default é undefined). Informações de sessão e outros dados que você pode precisar passar com cada requisição são comumente colocadas aqui.

url: A url default para ser usada para requisições no servidor (o default é undefined). Se o servidor receber todas as requisições através de uma URL, defini-la assim é mais fácil que defini-la em cada requisição.

defaultHeaders: Um objeto contendo cabeçalhos que serão adicionados a cada requisição feita por este objeto (default é undefined).

```
// Cabeçalhos padrão para passar em cada requisição

Ext.Ajax.defaultHeaders = {

    'Powered-By': 'Ext Core'

};
```

Ext.Ajax.request

Ext.Ajax.request é a função chamada para enviar e receber dados para o servidor via Ajax. Funções de sucesso e falha podem também ser definidas para manipular a resposta retornada pelo servidor. Observe que essas funções success/failure são assíncronas e serão chamadas de volta quando o servidor responder, enquanto isso estiver acontecendo sua página web continuará a operar.

```
Ext.Ajax.request({

   url: 'ajax_demo/sample.json',

   success: function(response, opts) {

      var obj = Ext.decode(response.responseText);

      console.dir(obj);

   },

   failure: function(response, opts) {

      console.log('falha no Server-side com o código de status ' +
response.status);

   }

});
```

Ext.Updater

Outro uso comum para Ajax é atualizar elementos dinamicamente em uma página sem recarregar tudo. O método request expõem uma configuração El que irá pegar a resposta de uma requisição e jogá-la no innerHTML de um elemento. Desenvolvedores podem também usar Ext.TaskMgr para definir uma tarefa recorrente para atualizar um elemento periodicamente.

Postando um formulário com Ajax

Use a configuração form para postar um formulário com Ext.Ajax.request

```
Ext.Ajax.request({

   url: 'ajax_demo/sample.json',

   form: 'myForm',

   success: function(response, opts) {

      var obj = Ext.decode(response.responseText);

      console.dir(obj);

   },

   failure: function(response, opts) {

      console.log('falha no servidor com o código de status ' +
response.status);

   }

});
```

Ext.DomQuery

O que é DomQuery?

DomQuery fornece um processamento seletor/xpath de alta performance. Ele trabalha em documentos HTML e XML (se o conteúdo de um nó é passado como parâmetro). DomQuery suporta a maioria dos seletores da especificação CSS3, junto com muitos seletores customizados e o XPath básico. Uma lista completa dos seletores da especificação CSS3 pode ser encontrada aqui.

Seleções múltiplas

Você pode selecionar múltiplos conjuntos de elementos com diferentes critérios dentro de um único conjunto de resultados.

```
// Combina todas as divs com a classe foo e todas as spans com a clase bar

Ext.select('div.foo, span.bar');
```

Seleção de raiz

Quando usando um seletor, é possível especificar um nó raiz opcional, se ele não for especificado irá usar como padrão o body do documento. Isto pode ser útil para aumentar a performance, uma vez que especificar uma raiz significa que a possibilidade de nós a serem verificados é menor.

```
Ext.get('myEl').select('div.foo');// esses são equivalentes

Ext.select('div.foo', true, 'myEl');// esses são equivalentes
```

Encadeamento de seleção

Seletores podem ser encadeados para combinar com múliplos critérios, que é útil se você precisa executar uma consulta complexa. Encadeamentos de atributos são processados em ordem.

```
// Combina uma div que com a classe de foo, que tem um atributo título bar, que é

// o primeiro filho do elemento pai imediato

Ext.select('div.foo[title=bar]:first');
```

Seletores de elementos

***** qualquer elemento
E um elemento com a tag E
E F todos os elementos descendentes de E que tem a tag F
E > F or E/F todos os elementos filhos diretos de E que têm tag F

E + F todos os elementos com a tag F que estão imediatamente precedidos por um elemento com a tag E
E ~ F todos os elementos com a tag F que estão precedidos por um elemento irmão com a tag E

```
// Combina todos os elementos div

Ext.select('div');

// Combina todos os elementos spans contidos dentro de uma div em qualquer nível

Ext.select('div span');

// Combina todos os elementos li com um ul como seu pai imediato

Ext.select('ul > li');
```

Seletores de atributos

E.foo tem a classe "foo"
E[foo] tem um atributo "foo"
E[foo=bar] tem um atributo "foo" que é igual a "bar"
E[foo^=bar] tem um atributo "foo" que começa com "bar"
E[foo$=bar] tem um atributo "foo" que termina com "bar"
E[foo*=bar] tem um atributo foo "foo" que contém a substring "bar"
E[foo%=2] tem um atributo "foo" que é divisível por 2
E[foo!=bar] tem um atributo "foo" que não é igual a "bar"

```
// Combina todos os elementos div com a classe news

Ext.select('div.news');

// Combina todos os elementos com um href que é http://extjs.com

Ext.select('a[href=http://extjs.com]');

// Combina todos os elementos img que possuem a tag alt

Ext.select('img[alt]');
```

Pseudo seletores

E:first-child E é o primeiro filho deste pai
E:last-child E é o último filho deste pai
E:nth-child(n) E é o n filho deste pai (1 é o padrão)
E:nth-child(odd) E é não é um filho deste pai
E:nth-child(even) E é mesmo um filho deste pai
E:only-child E é o único filho deste pai
E:checked E é um elemento que tem um atributo checked que é true (ex. um radio ou checkbox)
E:first o primeiro E no conjunto de resultados

E:last o último E no conjunto de resultados
E:nth(n) o n E no conjunto de resultados (Baseado em 1)
E:odd atalho para :nth-child(odd)
E:even atalho para :nth-child(even)
E:contains(foo) o innerHTML de E contém a substring "foo"
E:nodeValue(foo) E contém a textNode com um nodeValue que é igual a "foo"
E:not(S) um elemento E que não combina com um seletor simples S
E:has(S) um elemento E que tem um descendente que combina com um seletor simples S
E:next(S) um elemento E cujos próximos irmãos combinam com o seletor S
E:prev(S) um elemento cujos irmãos anteriores combinam com um seletor simples S

```
// Combina a primeira div com uma classe de código

Ext.select('div.code:first');

// Combina spans que caem em um mesmo índice

Ext.select('span:even');

// Combina todas as divs que os próximos irmãos estão em uma span com a classe header.

Ext.select('div:next(span.header));
```

Seletores de valores CSS

E{display=none} valor css "display" que é igual a "none"
E{display^=none} valor css"display" que começa com "none"
E{display$=none} valor css "display" que termina com "none"
E{display*=none} valor css "display" que contém a substring "none"
E{display%=2} valor css "display" que é divisível 2
E{display!=none} valor css "display" que não é igual a "none"

Gerando marcação dinâmica

DomHelper

DomHelper (abreviado para DH) é uma classe de utilidade usada para gerar dinamicamente marcações e trabalhar através de muitas questões cross-browser encontradas quando criamos marcações usando scripts DOM nativos. DH é suficientemente inteligente para usar HTML onde apropriado para otimizar a velocidade sem sacrificar a compatibilidade. Uma vez que Ext.DomHelper é um sigleton, todos os seus métodos podem ser acessados estaticamente sem instanciar uma nova instância da classe.

Ext.DomHelper é um singleton. Sendo assim você pode acessar todos os seus métodos estaticamente sem instanciar uma nova instância da classe.

markup

Mesmo como o depreciado createHtml

insertHtml

insertBefore

insertAfter

insertFirst

append

overwrite

Script DOM

```
var myEl = document.createElement('a');

myEl.href = 'http://www.yahoo.com/';

myEl.innerHTML = 'My Link';

myEl.setAttribute('target', '_blank');


var myDiv = document.createElement('div');

myDiv.id = 'my-div';


myDiv.appendChild(myEl);

document.body.appendChild(myDiv);
```

Ext.DomHelper

```
Ext.DomHelper.append(document.body, {
```

```
    id: 'my-div',
    cn: [{
        tag: 'a',
        href: 'http://www.yahoo.com/',
        html: 'My Link',
        target: '_blank'
    }]
});
```

Configurações DomHelper

Quando utilizado para gerar marcações dinamicamente, uma configuração DomHelper é usada para especificar qual marcação será criada em qual local da página. Uma configuração DomHelper é a representação de qualquer elemento arbitrário para criar na página.

Marcação

```
<a href="http://www.extjs.com">Ext JS</a>
```

Configuração DomHelper

```
{
    tag: 'a',
    href: 'http://www.extjs.com',
    html: 'Ext JS'
}
```

Template

Tpl, funções de formatação, a partir de métodos estáticos (importante de textarea)

Template funções membro

Adição e execução de funções membro de formatação

Acréscimos JavaScript

Sobre os acréscimos JavaScript

JavaScript é uma linguagem flexível e ela permite ao programador adicionar funções aos objetos base dentro do Javascript. A razão para fazer isso é porque há um número de funções que são extremamente úteis para usar em um objeto base, entretanto eles são ou não implementadas no todo, ou somente implementadas em certos navegadores. Quando usando múltiplas bibliotecas, elas podem ambas implementar funções em objetos base javascript. Esta sobreposição tem um enorme potencial para causar conflitos quando se usa estas bibliotecas juntas. Por esta razão, Ext adiciona somente algumas funções necessárias nos objetos base. Aqui está uma comparação interessante sobre diferentes bibliotecas e seus usos de acréscimos javascript: Framework Scanner.

Acréscimos em funções

As seguintes funções foram adicionadas à função prototype (observe que createSequence e createInterceptor não foram incluídas):

createCallback

Cria uma função de chamada de retorno que pode passar uma série de argumentos. Esses argumentos podem ser dinâmicos e definidos em tempo de execução. Esta função ignora o escopo, se ele é necessário você deve usar createDelegate. createCallback passa todos os argumentos passados para ela mesmo para a função de chamada de retorno.

```
var sayHello = function(firstName, lastName){

    alert('Hello ' + firstName + ' ' + lastName);

};

Ext.get('myButton').on('click',         sayHello.createCallback('John',
'Smith');
```

createDelegate

Está é similar a createCallback, entretanto é ligeiramente mais poderosa. Ela permite que você especifique o escopo em que a função de retorno será executada. Ela também dá mais controle sobre os argumentos que são passados para a função de retorno, especialmente se hpa argumentos existentes para o evento. O primeiro parâmetro é o escopo. O segundo parâmetro é um array de argumentos passados. O terceiro parâmetro permite que você controle como os argumentos serão passados. Se true, significa adicionar os argumentos ao final de qualquer argumento já passado (digamos no caso de um objeto de evento). Se false, significa somente passar o array de argumentos, nada mais. Se ele for um inteiro, os argumentos serão inseridos no índice específico.

```
var sayHello = function(firstName, lastName, e){

    alert('Hello ' + firstName + ' ' + lastName);
```

```
};

Ext.get('myButton').on(

   'click',

   sayHello.createDelegate(this, ['John', 'Smith'],

   //0 indica que nós queremos inserir nossos argumentos antes de qualquer outro.

   0

);
```

defer

defer permite que você especifique um período de tempo a esperar antes de executar uma função. O primeiro argumento é a quantidade de tempo a esperar em milisegundos. O segundo argumento é o escopo no qual a função será executada.

```
var whatsTheTime = function(){

    alert(new Date());

};

whatsTheTime.defer(3000); //espera 3 segundos antes de ser executado.
```

Acréssimos em Array

Os seguintes métodos foram adicionados ao prototype de Array, se e somente se eles já não tiverem sido implementados pelo navegador:

indexOf

Encontra o primeiro índice no array que combina com o parâmetro passado. Se não encontrar no array, -1 é retornado.

```
var idx = [1, 2, 3, 4, 5].indexOf(3); //Retorna 2.
```

remove

Remove a primeira instância do argumento passado do array, se encontrado. Observe que esta função modifica o array.

```
var arr = [1, 2, 3, 4];

arr.remove(2);

var len = arr.length; // len agora é 3.
```

Acréscimos em String

A classe string teve adicionado um método simples format. Observe que ele pode entrar em conflito quando usando Ajax.NET.

format

Permite que você defina uma string simbolizada e passa um número arbitrário de argumentos para substituir os símbolos. Cada símbolo precisa ser único, e precisa incrementar no formato {0}. {1};

```
var s = String.format(

   'Hey {0} {1}', how are you?',

   'John',

   'Smith'

);

//{0} é substituído por John, {1} é substituído por Smith.
```

Utilidades

Sobre utilidades

Ext fornece uma série de funções de utilidade geral para trabalhar com Javascript e JSON. Elas variam em suas finalidades, mas todas têm um objetivo: tornar sua vida, como desenvolvedor, mais fácil.

apply e applyIf

apply

A função apply é usada para copiar as propriedades de um objeto literal em outro. O primeiro argumento é o objeto destino. O segundo argumento é o código de onde será copiado. Observe que todas as propriedades no código serão copiadas por cima, mesmo se elas existirem no objeto destino. Também, o objeto destino será modificado como um resultado da chamada do método.

```
var person = {
    name: 'John Smith',
    age: 30
};

Ext.apply(person, {
    hobby: 'Coding',
    city: 'London'
}); // person literal agora contém hobby também city
```

applyIf

Esta função é muito similar a apply. A única diferença é que applyIf somente copia propriedades que não existem no objeto destino. Se elas já existirem em ambos, o objeto destino leva precedência e a propriedade não é copiada.

```
var person = {
    name: 'John Smith',
    age: 30,
    hobby: 'Rock Band'
};
```

```
Ext.applyIf(person, {
    hobby: 'Coding',
    city: 'London'
}); // hobby não é copiado por cima
```

Url Encoding/Decoding

Esses métodos são úteis para transformar dados JSON em um formato que pode ser transmitido como parte de uma string GET e virse versa.

urlEncode

Converte um objeto literal em uma string que pode ser passado como um parâmetro usando uma requisição GET. A string irá ser no formato chave1=valor1&chave2=valor2.....

```
var params = {
    foo: 'value1',
    bar: 100
};

var s = Ext.encode(params); // s é agora foo=value1&bar=100
```

urlDecode

Este método é o oposto de urlEncode. Ela recebe uma string no formato de chave1=valor1&chave2=valor2 e a converte em um objeto JavaScript.

```
var s = 'foo=value1&bar=100';
var o = Ext.decode(s); // o agora contém 2 propriedades, foo e bar.
alert(o.bar);
```

Manipulação de Array

A Ext Core fornece métodos para trabalhar com arrays e outras coleções dentro de javascript.

each

O método each permite que você itere sobre um array ou outras coleções (incluindo Nodelists ou CompositeElements). Each recebe uma função como argumento. Esta função é chamada para cada membro da coleção. Retornando true em qualquer ponto para a iteração.

```
Ext.each([1, 2, 3, 4, 5], function(num){
    alert(num);
});
```

toArray

Este método converte uma coleção iterável (ou elemento simples) em um array próprio de javascript.

```
var arr1 = Ext.toArray(1); // arr1 = [1];
// arr2 agora contém um array de elementos Ext.
var arr2 = Ext.toArray(Ext.select('div'));
```

JSON

JSON significa JavaScript Object Notation. Ele é usado como formato de troca de dados, onde os dados são muito similares a objetos literais javascript. Quando enviando e recuperando dados do servidor, ele é necessário para converter dados para e de sua forma nativa javascript. Essas funções de auxílio te ajudam a fazer isso. Mais informações sobre JSON podem ser encontradas em json.org.

encode

Recebe um objeto javascript ou array e o codifica como uma string de representação do JSON que pode ser passado para um código externo.

```
var s = Ext.encode({
    foo: 1,
    bar: 2
}); //s agora contém '{foo=1,bar=2}'
```

decode

O oposto de encode, esta é usada para converter uma string de representação de JSON em um objeto javascript. Este método é frequentemente chamado durante a leitura da resposta de uma chamada de retorno Ajax.

```
var s = '{foo=1,bar=2}';
var o = Ext.decode(s); o agora é um objeto com 2 propriedades, foo e bar.
```

Detecção de Navegador e SO

JavaScript

Ext oferece um número de recursos de detecção de navegadores que permitem aos desenvolvedores trabalharem em cima de questões de implementação devido a diferenças entre a maioria dos navegadores. Ext fornece detecção em javascript e css, para permitir uma maior funcionalidade em ambas essas áreas.

As seguintes detecções estão disponíveis para javascript:

Internet Explorer - Ext.isIE, Ext.isIE6, Ext.isIE7, Ext.isIE8
Firefox - Ext.isGecko, Ext.isGecko2, Ext.isGecko3
Opera - Ext.isOpera
Chrome - Ext.isChrome
Safari - Ext.isSafari, Ext.isSafari2, Ext.isSafari3
WebKit - Ext.isWebKit
Operating Systems - Ext.isLinux, Ext.isWindows, Ext.isMac

```
if(Ext.isIE){

    // faz o código do navegador específico aqui

}
```

CSS

Um mecanismo similar é aplicado no CSS, vários nomes de classes são adicionados ao elemento raiz e o corpo depende do ambiente operacional atual. Isto permite a fácil definição do estilo de regras para os navegadores. Se no modo estrito, ext-strict é adicionado à raiz. O resto desses são adicionados ao corpo quando apropriado:

.ext-ie, .ext-ie6, .ext-ie7, .ext-ie8
.ext-gecko, .ext-gecko2, .ext-gecko3
.ext-opera
.ext-safari
.ext-chrome
.ext-mac, .ext-linux

```
/* Quando no modo estrito e usando safari, muda o font-size. */

.ext-strict .ext-safari .sample-item{

    font-size: 20px;

}
```

Detecção de tipos

Uma vez que JavaScript é uma linguagem fracamente tipada, é frequentemente necessário interrogar as variávies para recuperar seus tipos. Ext fornece uma série de métodos para ajudar nesta tarefa:

isEmpty

Verifica se o valor passado é vazio, que inclui undefined, null ou uma string vazia.

```
alert(Ext.isEmpty(''));
```

isArray

Verifica se o valor passado é um array.

```
alert(Ext.isArray([1, 2, 3]));
```

isObject

Verifica se o valor passado é um objeto.

```
alert(Ext.isObject({}));
```

isFunction

Verifica se o valor passado é uma função.

```
alert(Ext.isFunction(function(){

}));
```

Miscelânia

id

Retorna uma string que contém um identificador único. Qualquer chamada a Ext.id() irá garantir o retorno de um novo id que não foi usado. O primeiro parâmetro é um elemento opcional, ao qual o id será atribuído. O segundo parâmetro opcional é um prefixo para o id.

```
var s = Ext.id(null, 'prefix'); // Nenhum elemento espeificado aqui

var s = Ext.id(Ext.get(document.body)); // atribui o id a um elemento
```

Execução de código temporizada

TaskMgr e TaskRunner

A classe TaskRunner é usada para executar uma função em um intervalo especificado. Esta é útil quando se está fazendo funções de sondagem, por exemplo quando queremos recarregar o conteúdo Ajax a cada 30 segundos. O objeto TaskMgr é uma instância singleton de TaskRunner, ele pode ser usado para acesso rápido a TaskRunner.

```
var stop = false;

var task = {

    run: function(){

        if(!stop){

            alert(new Date());

        }else{

            runner.stop(task);  // nós paramos a tarefa aqui se necessário.

        }

    },

    interval: 30000 // a cada 30 segundos

};

var runner = new Ext.util.TaskRunner();

runner.start(task);


// Usando TaskMgr

Ext.TaskMgr.start({

    run: function(){

    },

    interval: 1000

});
```

DelayedTask

A classe DelayedTask fornece uma maneira conveniente de "armazenar" a execução de um método. Quando chamada, a tarefa irá esperar um período de tempo especificado antes de executar. Se durante o período de tempo, a tarefa é chamada novamente, a chamada original será cancelada. Isto continua assim que a função é chamada somente uma única vez

para cada iteração. Este método é especialmente útil para coisas como detectar se o usuário finalizou a edição de um campo de texto.

```
var task = new Ext.util.DelayedTask(function(){

    alert(Ext.getDom('myInputField').value.length);

});

// Espera 500ms ante de chamar nossa função. Se o usuário pressionar outra tecla

// durante esses 500ms, ela será cancelada e nos iremos esperar outros 500ms.

Ext.get('myInputField').on('keypress', function(){

    task.delay(500);

});
```

Observe que nós estamos usando um DelayedTask aqui para ilustrar um ponto. A opção de configuração armazenada para addListener/on irá também configurar delayedtask para armazenar eventos.

Sistema de Classes

Classes JavaScript

JavaScript é baseado e protótipos, significando que ele se difere de uma classe típica baseada em linguagens. Em javascript, a criação de uma nova classe é feita pela modificação do protótipo de um objeto. Ext fornece um número de funções que tornam simples criar e trabalhar com classes. Há uma boa discussão em várias funções javascript baseadas em heranças aqui.

Ext.extend

Ext fornece uma série de funções para estender ou sobrescrever as classes javascript existentes. Isto siginifica que você pode adicionar comportamentos e criar suas próprias classes, ou sobrescrever o comportamento de uma seleção de poucas funções para servir às suas necessidades.

extend

Cria uma nova definição de classe. O primeiro argumento é a classe a estender. O segundo parâmetro é a lista de propriedades/funções a adicionar ao protótipo do objeto. Quando se usa extend, Ext mantém uma referência para a superclasse, como mostrado no segundo exemplo.

```
Person = Ext.extend(Object, {

    constructor: function(first, last){

        this.firstName = first;

        this.lastName = last;

    }


    getName: function(){

        return this.firstName + ' ' + this.lastName;

    }

});


Developer = Ext.extend(Person, {

    getName: function(){

        if(this.isCoding){

            return 'Go Away!';
```

```
        }else{
            // Acessa o método getName da superclasse
            return Developer.superclass.getName.call(this);
        }
    }
});


var p = new Person('John', 'Smith');
alert(p.getName());
```

override

Similar a extend, entretanto override não cria uma nova definição de classe. Em vez disso, somente modifica o comportamento da classe existente. O primeiro parâmetro é a classe a sobrescrever, o segundo parâmetro é a lista de propriedades/funções para adicionar o outro protótipo de objeto.

```
// Assuma que nós tempos declarado nossa classe Person acima.
Ext.override(Person, {
    getName: function(){
        // Sobrescreve o comportamento, retorna o último nome mais o primeiro.
        return this.lastName + ' ' + this.firstName;
    }
});
```

Coisas que vão no protótipo são compartilhadas

Quando colocamos itens no protótipo de do protótipo, elas serão compartilhadas entre todas as instâncias da classe. A menos que você especificamente queira fazer isso, você não deve colocar tipos "primitivos" dentro da definição do protótipo.

```
MyClass = Ext.extend(Object, {
    // Este objeto literal é agora compartilhado entre toda as instâncias de MyClass.
    baseParams: {},
```

```
    foo: function(){
        this.baseParams.bar = 'baz';
    }
});


Ext.onReady(function(){


    var a = new MyClass();
    var b = new MyClass();


    a.foo();
    // Modificando baseParams em a afeta baseParams em b.
    console.log(b.baseParams);
});
```

Singletons

Caso contrário conhecido como o módulo de padrão de projeto este padrão permite que você crie variávies ou métodos JS privados através do uso inteligente de encerramentos. O singleton é um bom padrão para usar quando você tem uma classe de métodos estáticos, ou você tem uma classe que será usada somente uma vez. Um bom candidato para um singleton é o ponto de entrada em sua aplicação.

```
MyApp = function(){
    var data; data é privado e não pode ser acessado de fora.
    return {
        init: function(){
            // Inicializa a aplicação aqui
        },


        getData: function(){
            return data;
        }
```

```
    };

}();

Ext.onReady(MyApp.init, MyApp);
```

Ext.util.Observable

O padrão Observable (ou assinante) é usado para dissociar objetos que necessitam saber detalhes sobre o estado de outros objetos. Isto é feito pelo uso de eventos. Quando o estado de um sujeito muda, o sujeito irá acionar um evento. Assinantes deste evento particular em um sujeito serão então notificados que a mudança de estado ocorreu. Muitas classes Ext estendem Observable para permitir um modelo de programação flexível e dissociado. Você pode facilmente criar uma classe que aciona eventos customizados:

```
var MyClass = Ext.extend(Ext.util.Observable, {

    constructor: function(config){

        this.addEvents('datachanged'); // especifique os eventos que
nós estamos indo acionar

        MyClass.constructor.call(this, config);

    },


    update: function(){

        // faça alguma transformação de dados aqui

        // quando acionado um evento, nós podemos especificar que

        //parâmetros são passados aos assinantes

        this.fireEvent('datachanged', this, this.data.length);

    }

});


// Assinar um evento

var c = new MyClass();

c.on('datachanged', function(obj, num){

    // reagir aos dados alterados do evento.

});
```

Namespaces

Namespaces são úteis para organizar seu código, eles fonecem 2 benefícios principais. O primeiro é que você pode usá-los para evitar a poluição do namespace global com objetos, que são geralmente considerados indesejados. Ext, por exemplo possui somente um único objeto global (o objeto Ext). Está é uma boa prática colocar qualquer classe dentro de um namespace, o mais comum é o nome da sua companhia ou o nome da sua aplicação. A outra vantagem é que esta ajuda a manter seu código organizado, você pode agrupar juntos classes similares ou co-dependentes no mesmo namespace, que ajuda a especificar sua intenção a outros desenvolvedores.

```
// Os seguindes são equivalentes, o último é preferido.

Ext.namespace(

   'MyCompany',

   'MyCompany.Application',

   'MyCompany.Application.Reports'

);

Ext.namespace('MyCompany.Application.Reports');
```