

# Ext 2.0 – Visão Geral

Traduzido por:

Otávio Augusto Rodrigues Fernandes

[otavio@neton.com.br](mailto:otavio@neton.com.br)

## Conteúdo

Conteúdo.....	2
Sobre este documento.....	3
Introdução.....	4
Resumo das principais mudanças.....	4
Observações adicionais.....	6
Modelo Component.....	7
Visão geral de Component.....	7
Ciclo de vida do componente.....	9
<b>Inicialização</b> .....	9
<b>Renderização</b> .....	11
<b>Destruição</b> .....	12
XTypes de Componentes.....	14
BoxComponent.....	15
Modelo Container.....	15
Container.....	15
Painel.....	16
Window.....	16
ViewPort.....	16
Layouts.....	17
Introdução.....	17
Gerenciadores de layout.....	19
Grid.....	21
Visão geral.....	21
Uma observação sobre o travamento de colunas.....	22
XTemplate.....	22
DataView.....	23
Outros novos componentes.....	24
Action.....	24
CycleButton.....	24
Hidden(field).....	25
ProgressBar.....	25
TimeField.....	25

## Sobre este documento

Este documento é uma tradução livre de [Ext 2 Overview](#), escrito originalmente por Brian Moeskau e publicado no site oficial de [ExtJS](#). O mesmo foi traduzido por Otávio Augusto Rodrigues Fernandes com o objetivo de auxiliar os desenvolvedores da [Net ON – Soluções Tecnológicas](#) no desenvolvimento de aplicações com esse framework. Visto que é um documento essencial a todo e qualquer desenvolvedor ExtJS, o mesmo está sendo disponibilizado gratuitamente para toda a comunidade.

Sugestões e melhorias são bem vindas e devem ser enviadas para [otavio@neton.com.br](mailto:otavio@neton.com.br).

### Introdução

**Bem vindo ao Ext 2.0.** Nas seções seguintes você irá aprender sobre todas as mudanças mais importantes da versão 2.0 de Ext. Você irá aprender, em alto nível, que novas funcionalidades existem e o que você poderá fazer com elas. Entretanto, como uma visão geral, este guia não irá discutir alguns dos detalhes que você precisará para começar escrever suas próprias aplicações Ext 2.0. Para informações adicionais, aqui estão alguns recursos completos:

- [Ext 1.x to 2.0 Migration Guide](#)
- [Ext 2.0 API Reference](#)
- [Ext 2.0 Examples](#)
- [Ext 2.0 Change Log](#)
- [Ext Community Forums](#)

### Resumo das principais mudanças

Aqui está um resumo em alto nível do que é novo na versão 2.0. Por favor, observe que ocorreram inúmeras pequenas melhorias, correções de bug e outras mudanças em todo o framework da versão 1.x para a 2.0. Seria impossível listar cada coisa, por esse motivo, esta visão geral está focada nas principais áreas de mudança, que a arquitetura tenha passado, ou onde haja alguma área com funcionalidade inteiramente nova. Cada item será explanado em detalhes nas seções seguintes a este resumo.

### Modelo de Componente

Apesar da existência das classes `Component` e `BoxComponent` na versão 1.x, elas não eram completamente integradas ao framework. Na versão 2.0, ambas as classes foram melhoradas, e agora formam a base de todos os principais componentes. Ainda que essas classes sejam destinadas a serem,

## Ext 2.0 - Visão Geral - Tradução

principalmente invisíveis ao desenvolvedor, um entendimento do ciclo de vida de um componente na versão 2.0 é fundamental para elevar suas habilidades em Ext 2.0 para o próximo nível.

### Modelo de Container

Há agora muitas classes fundamentais disponíveis para construir widgets e layouts que podem conter outros componentes. Container fornece o framework fundamental para o agrupamento e layout dos componentes, e é essencial para a totalidade do visual do framework Ext. Panel estende Container para fornecer a base de funcionalidade específica da aplicação UI (user interface) e é provavelmente a mais importante classe na hierarquia de container. Window é um tipo especial de Panel que habilita o verdadeiro estilo desktop nas aplicações web, e Viewport é um container de utilidade especificamente projetado para implementar aplicações web em full-browser-window.

### Layouts

Na versão 1.x, Layout era centrado em torno do BorderLayout e de suas respectivas classes. Na versão 2.0, toda a arquitetura do layout foi construída no novo Container e classes layout foram criadas. BorderLayout agora foi unido a nove estilos de layouts adicionais, e a hierarquia de classes foi redesenhada para permitir a máxima extensibilidade. Layouts são também completamente gerenciados na versão 2.0, removendo algumas complexidades que os desenvolvedores encontravam ao implementar layouts complexos na versão 1.x.

### Grids

O componente grid sempre foi uma das peças centrais de Ext, e com a versão 2.0 sua evolução prossegue. Novo nesta versão, está com a interface de usuário ainda mais trabalhada, performance melhorada, agrupamento de

## Ext 2.0 - Visão Geral - Tradução

linhas, linhas de resumo, exemplos de plugins de personalização fornecendo linhas expansíveis e linhas com números muito maiores.

### **XTemplate**

A classe Template na versão 1.0 funciona muito bem para templates simples, mas carece de alguns recursos chaves para criar saídas mais avançadas. Na versão 2.0, a nova classe XTemplate, que foi adicionada, fornece sub-templates, processamento de arrays, execução de código inline, lógica condicional e muito mais.

### **DataView**

Na versão 1.x, a classe View fornece moldes de vinculação de dados para gerar views UI personalizadas dos dados. O JsonView foi uma classe de ajuda para facilitar a vinculação de um template a um dado JSON. Na versão 2.0 as capacidades da View foram elevadas para o próximo nível com o DataView, que estende BoxComponent para facilitar sua adição a layouts e também suportar a nova classe XTemplate para um processamento mais poderoso do template.

### **Outros novos componentes**

Muitos componentes e widgets foram adicionados na versão 2.0, incluindo, Action, CycleButton, Hidden(campo), ProgressBar, e TimeField.

### **Observações adicionais**

#### **Temas**

O suporte a temas na versão 2.0 foi simplificado. Ext suportava quatro diferentes temas na versão 1.x este número foi reduzido para dois (“Ext Blue” e Gray). Temas personalizados podem facilmente ser adicionados usando a

## **Ext 2.0 - Visão Geral - Tradução**

folha de estilos do tema Gray como um exemplo de implementação, e, um site de temas da comunidade está sendo planejado. Ainda que não seja uma mudança no código da API, é uma notável mudança que deve ser mencionada.

### **Mudanças compatibilidade**

Infelizmente há algumas mudanças na versão 2.0 que não tornaram inviável manter compatibilidade com a versão 1.x. Porque os componentes subjacentes e os modelos de renderização foram modificados consideravelmente. Alguns componentes existentes tiveram que ser reescritos de uma maneira que os tornou incompatíveis com seus antecessores na versão 1.x. Nós fornecemos um Guia de migração da versão 1.x para a versão 2.0 no site, que esperamos aliviar os encargos de atualizar uma aplicação da versão 1.x.

## **Modelo Component**

### **Visão geral de Component**

Um dos objetivos, com a versão 2.0, foi fornecer ainda mais blocos de construção básicos do que nunca. A classe Component foi originalmente introduzida na versão 1.x, mas não foi completamente integrada ao framework. Na versão 2.0, as capacidades de Component foram ainda ampliadas e melhoradas, fazendo dela uma das classes fundamentais na arquitetura global. Component agora fornece um modelo unificado para a criação, renderização, captura de eventos, gerenciamento de estado e destruição de componentes, e cada componente de Ext que requer esses recursos, agora estendem Component. Aqui estão alguns recursos chaves de Component na versão 2.0.

### **Encadeamento explícito de construtor e sobrecarga**

Component fornece um construtor base que irá aplicar alguma configuração passada em qualquer subclasse, e a função `initComponent` pode ser utilizada em sua totalidade através da herança encadeada para fornecer lógicas de construtores customizados em cada passo da hierarquia.

### **Renderização gerenciada**

Na versão 2.0, cada componente automaticamente suporta renderização lazy (por demanda), e o pipeline de renderização é gerenciado automaticamente, mesmo assim, você ainda pode ter uma flexibilidade final para personalizar o processo de renderização através de eventos antes e depois da renderização.

### **Destruição gerenciada**

Cada componente também possui uma função destrutora, e Ext gerencia automaticamente o coletor de lixo e destruição de componentes quando eles não são mais necessários. Claro, a destruição também fornece eventos antes e depois de sua execução que podem ser customizados quando necessário.

### **Gerenciamento automático de estado**

Componentes agora têm a funcionalidade built-in para configurar e recuperar um estado, e qualquer componente Ext suporta automaticamente o gerenciamento de estado quando o nível de `StateManager` global foi iniciado com um `Provider` de estado.

### **Interface consistente para o comportamento básico de componente**

A maioria dos comportamentos fundamentais que podem ser aplicados a qualquer componente, tais como esconder, exibir, desabilitar, habilitar são fornecidos via `Component`, e esses podem ser sobrescritos ou customizados por qualquer classe conforme necessário.

### Avaliação via ComponentMgr

Cada componente Ext fica registrado no ComponentMgr na sua criação, dessa forma, você pode recuperar qualquer componente a qualquer momento, simplesmente através da chamada `Ext.getCmp('id')`.

### Suporte a plugin

Qualquer componente pode agora ser estendido através do uso de plugins. Um plugin é uma classe com um método **init** que aceita um parâmetro simples do tipo `Ext.Component`. Plugins podem ser adicionados em qualquer componente através da opção de configuração de plugins. Quando um componente é criado, se algum plugin está disponível, o componente irá chamar o método `init` em cada plugin, passando uma referência de si mesmo. Cada plugin pode então chamar métodos ou responder a eventos conforme necessário para fornecer sua funcionalidade.

### Ciclo de vida do componente

Em geral, a arquitetura de Component na versão 2.0 irá trabalhar perfeitamente. Ela foi projetada para pegar a maioria dos eventos dos componentes de forma transparente ao desenvolvedor final. Entretanto, haverá momentos em que algumas coisas precisarão ser customizadas, ou um componente precisará ser estendido. Isto é, quando um conhecimento profundo do ciclo de vida do componente será muito útil. A seguir estão os estágios mais importantes no ciclo de vida de cada classe baseada em Component:

#### Inicialização

##### 1. A configuração do objeto é aplicada

As classes que estendem Component não precisam fornecer (e normalmente não devem) um construtor separado. O construtor de Component não irá

## Ext 2.0 - Visão Geral - Tradução

somente aplicar qualquer configuração passada para ele pela subclasse, ele também fornecerá todos os passos seguintes.

### **2. Os eventos base do componente são criados**

Esses são eventos que podem ser ativados por qualquer Component, e eles são enable, disable, beforeshow, show, beforehide, hide, beforerender, render e destroy.

### **3. O componente é registrado em ComponentMgr**

Como tal, será sempre disponível através de Ext.getCmp

### **4. O método initComponents é chamado**

Este é o passo de inicialização mais importante para as subclasses, visto que ele é um molde de método, destinado a ser implementado por cada subclasse para fornecer qualquer construtor lógico necessário. A classe que está sendo criada é chamada primeiro, e de cada classe na hierarquia superior de Component espera-se que chame initComponents da superclasse. Este método faz com que seja fácil implementar, e, se necessário, sobrescrever a lógica do construtor em qualquer passo da hierarquia.

### **5. O estado é aplicado (se aplicável)**

Se o componente for de estado consciente, este estado será recarregado se for aplicável.

### **6. Plugins são carregados (se aplicável)**

Se este componente tiver qualquer plugin especificado na configuração, ele será inicializado nesta hora.

## 7. O componente é renderizado (se aplicável)

O componente é renderizado imediatamente se `renderTo` ou `applyTo` for fornecido na configuração, senão a renderização é adiada até que o componente seja realmente exibido ou que isso seja dito pelo seu container.

## Renderização

### 1. O evento `beforeRender` é ativado

Este é um evento cancelável, dando a qualquer manipulador a capacidade de impedir a renderização do Component, se necessário.

### 2. O container é setado

Se o container não for especificado, o parent deste elemento será definido como seu container.

### 3. O método `onRender` é chamado

Este é o passo mais importante de renderização para as subclasses, visto que, este é um método molde, destinado a ser implementado por cada uma das subclasses para fornecer a lógica de renderização necessária. A classe que está sendo criada é chamada primeiro e espera-se de cada classe na hierarquia superior a chamada ao método `onRender` da superclasse. Este método faz com que seja fácil implementar, e, se necessário, sobrescrever a lógica de renderização em qualquer passo da hierarquia.

### 4. O componente é “desescondido”

Por default muitos componentes são escondidos usando a classe de estilos CSS especial “`x-hidden`”. Se o valor da configuração `autoShow` é `true`, qualquer classe “`hide`” será removida do componente.

### **5. Classes css personalizadas / ou estilos são aplicados**

Todas as subclasses de Component suportam uma propriedade de configuração especial de cls e style que são classes CSS especiais e as regras que respectivamente serão aplicadas ao elemento subjacente de Component. Adicionando cls ao componente e usar as regras padrões CSS de herança é o método adequado para personalizar visualmente um componente e suas partes constituintes.

### **6. O evento render é acionado**

Está é uma notificação simples de que o componente foi renderizado com sucesso até este momento.

### **7. O método afterRender é chamado**

Este é outro método molde que pode ser implementado ou sobrescrito para fornecer alguma lógica especial pós-renderização, que seja necessária. Cada subclasse chama o método afterRender da sua superclasse.

### **8. O componente é escondido e / ou desabilitado (se aplicável)**

Os valores da configuração hidden e disable são aplicados neste ponto.

### **9. Alguns eventos de especificação de estado são inicializados (se aplicável)**

Os componentes de estado conscientes podem declarar eventos especiais que são específicos para carregar e salvar o estado. Se fornecidos, qualquer um desses eventos será acionado.

## **Destruição**

### **1. O evento beforedestroy é acionado**

Este é um evento cancelável, dando a qualquer manipulador a habilidade de impedir que um componente seja destruído.

## **2. O método beforeDestroy é chamado**

Este é outro molde de método que pode ser implementado ou sobrescrito para fornecer qualquer lógica especial de pré-destruição, que seja necessária. Cada subclasse chama o método beforeDestroy da sua superclasse.

## **3. O Elemento e seus listeners são removidos**

Se o componente foi renderizado, seus listeners de eventos de elementos subjacentes são removidos e o elemento é então removido da DOM.

## **4. O método onDestroy é chamado**

Este é outro molde de método que pode ser implementado ou sobrescrito para fornecer alguma lógica especial no momento da destruição, que seja necessária. Cada subclasse chama o método onDestroy da sua superclasse. Observe que a classe Container (e qualquer de suas subclasses) fornece uma implementação default para onDestroy, que automaticamente em um laço de repetição chama o evento onDestroy de cada componente contido dentro dele.

## **5. O componente é “desregistrado” de ComponentMgr**

Ele não estará mais disponível através de Ext.getCmp

## **6. O evento destroy do componente é acionado**

Esta é uma notificação simples de que o componente foi destruído com sucesso neste ponto.

## **7. Os eventos listeners no componente são removidos**

O próprio componente pode ter eventos listeners separados dos seus elementos subjacentes. Se qualquer um existir, eles serão removidos.

## XTypes de Componentes

Um novo conceito é o na versão 2.0 é o de xtypes ou, tipos de componentes específicos de Ext. O tipo de xtypes disponíveis está resumido no cabeçalho da [API da classe Component](#). XTypes podem ser utilizados de maneira similar aos tipos de objetos JavaScript para dar uma olhada ou comparar componentes pelo tipo usando métodos como isXType e getXType. Você pode também listar a hierarquia de xtype de qualquer componente utilizando getXTypes.

Entretanto, o poder real do xtype está em como ele pode ser utilizado para otimizar a criação e renderização de componentes. Qualquer componente pode ser criado implicitamente com um objeto de configuração com um xtype especificado. Permitindo a ele ser declarado e passado para um canal de renderização sem realmente ser instanciado como um objeto. Não somente é renderizado posteriormente, como a real criação do objeto é também postergada, economizando memória e recursos até que ele realmente seja necessário. Em layouts encadeados complexos, que contém muitos componentes, isso pode fazer uma melhoria considerável na performance.

```
//Criação explícita de componentes contidos:
var panel = new Ext.Panel({
    ...
    items: [
        new Ext.Button({
            text: 'OK'
        })
    ]
});

//Criação implícita usando xtype:
var panel = new Ext.Panel({
    ...
    items: [{
        xtype: 'button',
        text: 'OK'
    }]
});
```

No primeiro exemplo, o botão sempre será criado imediatamente durante a inicialização do painel. Com muitos componentes adicionados, essa abordagem

## Ext 2.0 - Visão Geral - Tradução

pode deixar a renderização da página potencialmente mais lenta. No segundo exemplo, o botão não será criado ou renderizado até que o painel seja realmente mostrado no navegador. Se o painel jamais for mostrado (por exemplo, se ele é uma aba que permanece escondida) então o botão nunca será criado e dessa forma não irá consumir qualquer recurso de memória ou processamento.

### **BoxComponent**

Embora não sendo exclusivo da versão 2.0, BoxComponent é outra classe de fundamental, que estende Component, e fornece uma implementação de boxmodel consistente e cross-browser, que será visualmente renderizado e podem participar dos layouts. BoxComponent manipula de forma transparente o redimensionamento e reposicionamento, automaticamente, tratam qualquer diferença específica de navegador em padding, margens e bordas para produzir uma boxmodel consistente em todo e qualquer navegador compatível. Todas as classes container na versão 2.0 estendem BoxModel.

### **Modelo Container**

#### **Container**

Container é a fundação mais básica para qualquer componente que irá conter outros componentes. Ele fornece o layout e a lógica de renderização necessária para manipular o redimensionamento e encadeamento de outros componentes, e também fornece o mecanismo básico para adicionar consistência aos componentes do container. Container nunca deve ser usado diretamente, mas é destinado a ser usado como classe base para todos os componentes containers visuais.

### **Painel**

Painel é um container “trabalhador” na versão 2.0 e será o que você usará em 90% das tarefas em seu layout. Na sua forma mais básica, um painel pode ser uma caixa, completamente não visual, usada para construção de layouts. Entretanto, painel também fornece os blocos de construção básicos necessários em uma aplicação UI window, incluindo barras inferiores e de topo para adicionar barras de ferramentas, menus, um cabeçalho, rodapé e o corpo. Ele também fornece comportamentos expandir e encolher, e barra de botões, bem como uma variedade de botões pré-definidos para uma variedade de outras ferramentas. Painéis podem ser facilmente colocados dentro de um Container ou layout agora, e o layout e a lógica de renderização são completamente gerenciados.

As subclasses Panel a seguir são os principais widgets em Ext 2.0:

- GridPanel
- TabPanel
- TreePanel
- FormPanel

### **Window**

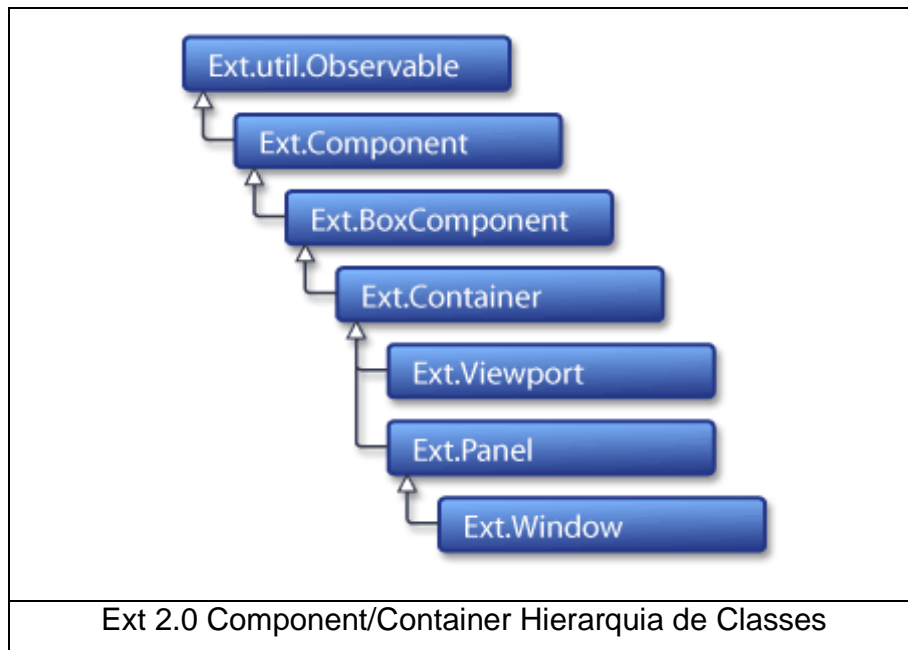
Window é um painel especializado que pode flutuar, ser minimizado e maximizado, restaurado, arrastado, etc. Ele foi projetado para ser uma classe de fundação básica para aplicações estilo desktop.

### **ViewPort**

O ViewPort é uma classe container muito útil, que pode renderizar-se automaticamente, ajustando-se ao corpo do documento, os tamanhos e sua própria dimensão se ajustam ao viewport do navegador. Este é um atalho

## Ext 2.0 - Visão Geral - Tradução

acessível para criar aplicações full-screen como o redimensionamento do navegador e o recalculamento são gerenciados por você automaticamente. Note que, ViewPort não pode ser renderizado em qualquer outro container que não seja o corpo do documento (document.body), e, como tal, você só pode utilizar uma instância de ViewPort por página.



## Layouts

### Introdução

Uma das áreas de melhoria, mais significantes na versão 2.0, é a facilidade e flexibilidade com que você pode criar sofisticados layouts de aplicação. Na versão 1.x o desenvolvimento de layout era centrado em torno do BorderLayout, Region e ContentPanel. Enquanto com o BorderLayout da versão 1.x era extremamente fácil gerar interfaces de usuário atrativas, ele não fornecia blocos de construção

## Ext 2.0 - Visão Geral - Tradução

suficientes para permitir, verdadeiramente, uma criação de layout customizada. A criação de layouts complexos, no entanto, requeria alguma codificação manual para lidar com barras de rolagem questões de encadeamento entre outras.

A versão 2.0 fornece um sistema de nível de gerenciamento de layout enterprise completamente renovado. Há agora 10 gerenciadores de layouts separados, fornecendo uma fundação para a construção de qualquer estilo possível de layout na aplicação. Layouts são também gerenciados por Ext, a fim de que, tamanho, posição, deslocamento e outros atributos simplesmente trabalhem como esperado. Você pode misturar e combinar diferentes containers, cada um com um layout diferente, encadeados em qualquer nível que você quiser, e um encaixe.

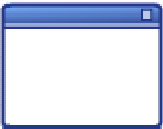
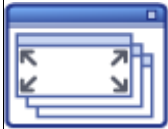


Layouts não são destinados para serem criados diretamente através da palavra chave **new**. Em vez disso, eles são criados e usados internamente pelas classes container. Os próprios containers não sabem nada sobre o layout – eles simplesmente delegam deveres ao layout, conforme a disposição da classe que foi especificada durante sua configuração.



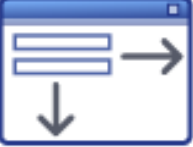



Em qualquer momento que você criar um container, você pode configurar seu estilo e configurar qualquer opção específica à classe layout através da propriedade `layoutConfig`. Por exemplo:

```
var panel = new Panel({
    title: 'My Accordion',
    layout: 'accordion', // O estilo do layout para
    // usar neste painel
    layoutConfig: {
        animate: true // valores específicos de
        // configuração do layout vão aqui
    }
    // opções adicionais do painel...
});
```

Cada classe layout suporta suas próprias opções de configuração específicas. Verifique os documentos de referência da API para cada layout, para mais detalhes.

## Gerenciadores de layout

 <p><b>ContainerLayout</b></p> <p>Esta é a classe base para todos os outros gerenciadores de layout, e é o layout padrão quando um layout específico não é definido. ContainerLayout não tem representação visual – ele simplesmente gerencia os itens armazenados, renderizando-os quando necessário e manipulando as tarefas básicas como armazenamento e redimensionamento. ContainerLayouts geralmente não devem ser criados diretamente, embora ele possa ser estendido para criar layouts customizados. Veja a API para referência.</p>	 <p><b>CardLayout</b></p> <p>CardLayout é um layout específico usado onde um container tem múltiplos elementos, mas somente um pode estar visível num dado momento. Mais comumente, este estilo de layout é usado para wizards, implementação de tabs customizadas, ou qualquer outro caso que requeira múltiplas páginas de informação mutuamente-exclusivas. Veja a API para referência.</p>
 <p><b>AbsoluteLayout</b></p> <p>Este é um layout muito simples que permite posicionar os itens armazenados precisamente nas coordenadas X/Y relativas ao container. Veja a API para referência.</p>	 <p><b>ColumnLayout</b></p> <p>Este é um estilo de layout para criar layouts estruturais no formato multi-coluna, onde, a largura de cada coluna pode ser especificada em pixels ou porcentagem, mas a altura, é permitido variar baseada no conteúdo. Veja a API para referência.</p>

 <p><b>AcordionLayout</b> O AcordionLayout contém um conjunto de painéis empilhados na vertical que podem ser expandidos e encolhidos para mostrar o conteúdo. Somente um painel pode abrir ao mesmo tempo. Veja a API para referência.</p>	 <p><b>FitLayout</b> Este é um estilo de layout simples que ajusta um simples item armazenado às exatas dimensões do seu container. Veja a API para referência.</p>
 <p><b>AnchorLayout</b> Este layout é para ancorar elementos aos lados relativos do seu container. Os elementos podem ser ancorados por porcentagem ou por compensações de bordas, e eles também suportam um layout virtual canvas, que pode ter diferentes dimensões em relação ao container. Veja a API para referência.</p>	 <p><b>FormLayout</b> O FormLayout é um layout muito útil especificamente projetado para criar formulários de entrada de dados. Observe que, em geral, você provavelmente vai querer usar FormPanel ao invés de um painel regular com layout: 'form' desde que FormPanel também fornece manipulação de submissão automática. Veja a API para referência.</p>
 <p><b>Border Layout</b> Este é exatamente o mesmo estilo de layout de BorderLayout como na versão 1.x – regiões de layouts suportam encadeamento, painéis abrindo e fechando com separadores de regiões. Este é o estilo de layout mais comum para as interfaces de usuário primária nas típicas aplicações de negócio. Veja a API para referência.</p>	 <p><b>TableLayout</b> Este estilo de layout gera marcações de tabelas padrões HTML que suportam colunas e linhas. Veja a API para referência.</p>

## Grid

### Visão geral

O componente de interface grid é implementado atualmente pela classe GridView, e na versão 2.0 GridView foi grandemente melhorada. Aqui estão os mais importantes novos recursos na versão 2.0:

#### Melhoria de Performance

A estrutura subjacente e o código de renderização para o GridView foram completamente refatorados na versão 2.0, tendo como prioridade a performance. Como parte desta prioridade o recurso de travamento de coluna foi removido. (veja a próxima seção).

#### Melhoria da aparência e visual

Juntamente com os novos temas na versão 2.0, o grid recebeu uma interface elevada o que o tornou mais atrativo do que antes.

#### Nível simples de agrupamento de linhas

Linhas podem ser agrupadas em uma dada coluna, e reagrupadas pelo usuário dinamicamente.

#### Grupos de linhas de resumo

Cada grupo de linhas pode ter uma linha de resumo opcional para resumir os dados no grupo.

#### Suporte avançado a plugins

Na versão 2.0, o suporte a plugin em geral é novo. O GridView, em particular, usa uma nova arquitetura de plugin para um grande efeito, incluindo vários plugins pré-fabricados como exemplo. O plugin RowExpander fornece linhas

## Ext 2.0 - Visão Geral - Tradução

expansíveis/encolhíveis, e o plugin RowNumbers fornece um suporte simples à numeração de linhas.

### Uma observação sobre o travamento de colunas

Alguns podem perceber que o recurso de travamento de coluna, introduzido na versão 1.1, foi removido na versão 2.0 e não será mais suportado. O travamento de colunas, que útil para um pequeno subconjunto de usuários, não foi compatível com muitos dos recursos implementados no GridView na versão 2.0 (agrupamento, sumário, etc) e liderava a perda de performance para todos os dois devido à forma do grid ser renderizado a fim de suportar o travamento de colunas. Isto pode ser possível portando a GridView da versão anterior 1.x para a 2.0, ou aplicando o suporte a isso no GridView, mas isso não será feito oficialmente pelo time de Ext.

### XTemplate

XTemplate fornece uma variedade de tags embutidas e operadores especiais para suportar um processamento extremamente robusto de templates usando estruturas de dados complexas. Aqui está uma lista em alto nível dos recursos suportados – para detalhes completos e exemplos de utilização, por favor veja a documentação da API de XTemplate.

- Auto-preenchimento de arrays e escopo de comutação;
- Acesso ao objeto pai dentro do escopo do sub-template;
- Array com índice de itens variável;
- Operador básico de combinação no suporte de valores de dados;
- Auto-renderização de arrays simples (que não contém objetos nos valores);
- Lógica condicional básica usando **IF**;
- Habilidade para executar códigos arbitrários dentro do template;
- Suporte para template de configuração de propriedades;

## Ext 2.0 - Visão Geral - Tradução

- Métodos de template customizados através de objetos de configuração;

### **DataView**

Superficialmente, DataView é muito similar à classe View na versão 1.x. Ambas suportam a renderização de templates de dados, ambas estão vinculadas as data stores e ambas tem modelos de seleção e eventos embutidos. Entretanto, DataView está um imenso passo à frente, pois ele utiliza o poder completo da arquitetura da versão 2.0. Aqui estão as mudanças mais significativas:

#### **Estende de BoxComponent**

A classe View na versão 1.x estende de Observable, o que significa que enquanto ela trabalhava como um grande componente independente ela não tinha a capacidade de participar em layouts de outros componentes. DataView, por outro lado, estende BoxComponent. Como discutido acima, ele fornece muitas vantagens, incluindo um ciclo de vida de componente gerenciado e capacidades de layout de BoxComponent.

#### **Alavanca XTemplate**

A Classe View usava internamente uma classe Template, na versão 1.x, para seu processamento de template. Ela trabalhava perfeitamente para visões de layout relativamente simples, faltava o poder para manipular tarefas complexas de renderização. DataView usa a classe XTemplate da versão 2.0 para seu processamento de template. Como discutido anteriormente, ela fornece a mais poderosa máquina de template e habilita o DataView a renderizar dados complexos em uma interface de usuários customizada facilmente.

### Configuração de opções adicionais

DataView fornece mais funcionalidades, com as seguintes novas configurações:

- **itemSelector:** DataView requer um seletor DOMQuery para saber como identificar cada item na visão. Isso fornece uma maior flexibilidade e velocidade comparada à versão 1.x.
- **simpleSelect:** Esta é uma nova opção de modo de seleção que habilita a multi-seleção através de clicar em múltiplos itens sem requerer que o usuário trave Shift ou Ctrl.
- **overClass:** Uma classe CSS que é automaticamente aplicada a cada elemento no mouseover e removida no mouseout.
- **loadingText:** DataView suporta o mesmo padrão automático de máscara de carregamento, como outros componentes em Ext que são vinculados a data stores.

### Outros novos componentes

Muitos novos componentes e widgets interessantes foram adicionados na versão 2.0. Dê uma olhada na documentação da API para obter maiores detalhes do que cada componente pode oferecer.

#### Action

Action é uma parte de funcionalidade reutilizável que pode ser abstraída por qualquer componente particular, de modo que possa ser utilmente compartilhada entre os vários componentes.

#### CycleButton

## Ext 2.0 - Visão Geral - Tradução

Esta é uma implementação de um SplitButton especializado que contém um menu de CheckItens, o botão automaticamente gira a cada item clicado elevando o botão da mudança de evento (ou chamando a função changeHandler, fornecidos) para cada item de menu ativo.

### **Hidden(field)**

Este é um campo simples e conveniente que é renderizado como um campo de entrada hidden padrão HTML. Ele é silenciosamente usado para armazenar valores para o submit form, e, na versão 2.0, esses campos podem ser criados e manipulados como outros componentes de formulário Ext.

### **ProgressBar**

Como uma simples barra de progresso embutida na classe MessageBox na versão 1.x. Agora ela foi criada fora, em um componente separado, o que a melhorou bastante. Ela suporta dois modos diferentes (progresso manual e automático) e o seu estilo visual pode ser customizado.

### **TimeField**

Esta é uma simples implementação de um seletor de tempo suspenso.