

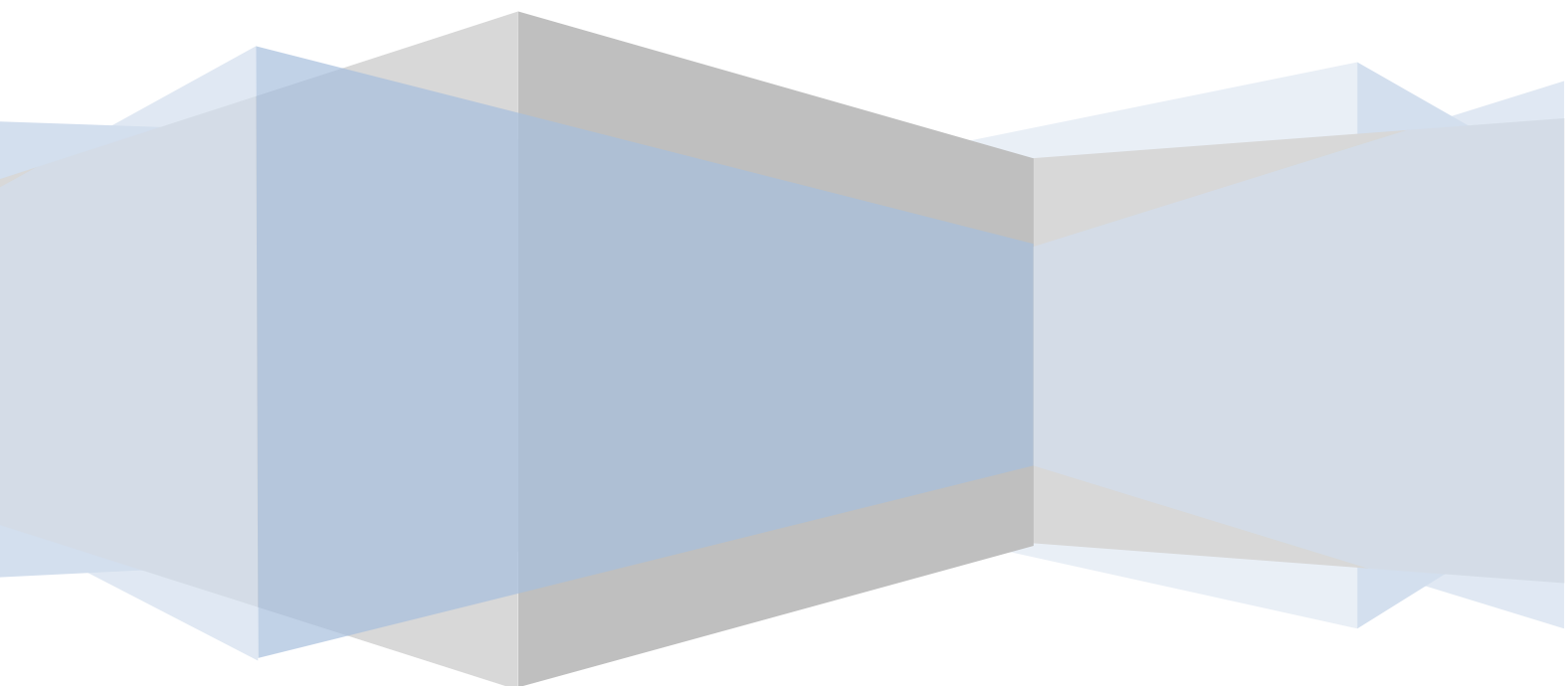
# Flex para desenvolvedores PHP

<http://corlan.org/flex-for-php-developers/>

**Por Mihai Corlan.**

**Tradução: Daniel Schmitz**

**Colaboradores:**



# Flex para desenvolvedores PHP

Eu tenho trabalhado com tecnologias relacionadas à web desde o final dos anos 90 e a minha primeira tecnologia *server side* foi PHP. Mais tarde trabalhei com ColdFusion e Java, mas sempre me considerei um programador PHP. Quando o Ajax surgiu, comecei a trabalhar com frameworks como *Prototype* e *script.aculo.us*, e comecei a criar os meus próprios frameworks.

No final de 2006, experimentei pela primeira vez o desenvolvimento Flex. Eu fiz uma espécie de curso intensivo, pois tinha de 4 a 6 semanas para criar uma aplicação de demonstração para a próxima versão do FDS (Flex Data Services, agora chamado de LiveCycle Data Services). Embora eu fosse novo no Flex e no FDS, o projeto correu bem, e realmente gostei do processo de desenvolvimento e aprendizagem.

No entanto, apesar de ter gostado, era diferente. Quero dizer que quando eu fiz o curso de desenvolvimento web com ColdFusion e Java, eu não sentia diferenças em relação ao PHP; era mais uma questão de encontrar a API correta e acostumar com as especificações da linguagem. Mais tarde, quando comecei a trabalhar com Ajax e DHTML, vi também que não havia diferenças. Você continuava criando a maior parte do *website* com as mesmas técnicas aprendidas e usava uma linguagem *server side*, jogando algum pedaço de código aqui ou ali (neste caso, alguns *widgets Ajax*).

Quando eu fiz o meu primeiro projeto web usando Flex, caramba, era completamente diferente. A separação clara entre o cliente e o servidor (lógica de negócio no lado do cliente além da lógica de negócio no lado do servidor), o cliente possuía uma tecnologia compilada ao invés de interpretada, duas línguas diferentes no lado do cliente, tudo isso pediu um pensamento diferente sobre o desenvolvimento web tradicional.

E esta é a minha razão de escrever este artigo. Eu quero compartilhar com vocês algumas coisas que são específicas para o Flex em relação ao PHP. Ao mesmo tempo, quero fazer uma introdução do Flex através de uma comparação com o PHP, sempre que esta comparação fizer sentido. Então, este artigo é para:

- Desenvolvedores PHP que querem aprender mais sobre Flex e Action Script 3, além de uma simples definição sobre o que são estas tecnologias
- Desenvolvedores PHP que já fizeram a sua primeira tentativa de codificação em uma aplicação Flex, e desejam ter uma melhor compreensão sobre o Flex.

O que não tem neste artigo? Não é a minha intenção de convertê-lo, nem de te convencer que o Flex é melhor que X ou Y. Eu acredito seriamente que existem diversos tipos de projetos e que eles podem e devem ser feitos com diferentes ferramentas. E quanto a performance, maximização de retorno em investimento e usabilidade são prioridades, não existe uma ferramenta universal para isso.

A propósito, este artigo não é um documento completo sobre o Flex ou Action Script 3. Há dezenas de livros que cobrem este assunto em centenas de páginas. Existem milhares de artigos sobre Flex. Minha intenção é fornecer informações suficientes para os temas mais importantes e, sempre que fizer sentido, relacionar estes conceitos com conceitos similares do PHP. Para manter o artigo útil, ele foi estruturado para não entrar em muitos detalhes. No final do artigo, vou dar uma breve introdução ao Adobe AIR e adicionar alguns recursos caso deseje obter maiores informações sobre este tópico.

Por fim, eu escolhi usar o Flex 3 na maioria dos exemplos. Existem algumas razões para esta escolha. Primeiramente, no momento em que escrevia este artigo, o Flex 4 estava na versão beta. Segundo, o Flex 4 é uma evolução do Flex, assim, a maioria dos assuntos abordados serão válidos para que sejam aplicados no Flex, com mínimas mudanças, se houver alguma. Em alguns casos, eu vou comentar estas diferenças. Quanto ao PHP, escolhi a versão 5.3 como referência. Agora que já falei tudo, vamos ver os assuntos abordados e, em seguida, vamos começar.

Sumário

## Conteúdo

Flex para desenvolvedores PHP .....	2
O que é Flex? .....	5
Flex: duas linguagens, um único framework.....	5
Porque você deve se preocupar com Flex .....	6
De um cliente leve para um cliente esperto/rico.....	8
Introdução à linguagem MXML.....	8
Misturando MXML e ActionScript 3.....	12
Estilos CSS.....	14
Modificando o código MXML em tempo de execução .....	16
Introdução à linguagem Action Script 3.....	17
Separando as Declarações .....	18
Os tipos de dados, variáveis e constantes .....	18
Funções e funções anônimas .....	22
OO: Classes e interfaces .....	24

Herança .....	28
Getters / Setters.....	30
Interfaces.....	32
Exceções.....	32
Convertendo e testando o tipo de um objeto .....	32
Escopo de variáveis .....	33
Matrizes (arrays) .....	34
Namespaces .....	35
Acessibilidade no <i>Namespace</i> .....	38
Trabalhando com XML .....	38
Dynamic Actionscript .....	39
Flex é assíncrono .....	40
DataBind, metada tags e reflection.....	41
Onde estão meus dados? Traga-os pra mim! .....	46
Autenticação de usuários em projetos Flex e PHP.....	48
Trabalhando em projetos Flex e PHP .....	49
Editores de texto para Flex e PHP .....	49
Flex Builder / Flash Builder e Eclipse PDT / Zend Studio.....	49
Depuração de aplicações FLEX.....	50
O que é Adobe Air .....	50
O que vem a seguir?.....	52
Para onde ir agora? .....	52
Tour de Flex.....	52
Livros .....	53
Websites.....	54

## O que é Flex?

A resposta mais simples é: Flex é apenas uma outra maneira de criar uma aplicação em Flash. Uma aplicação Flex é compilada em um arquivo SWF, que roda dentro do navegador através do Flash Player. Por que precisamos de uma outra maneira de criar aplicações em Flash? Tradicionalmente as aplicações em Flash foram criadas usando as ferramentas Flash. Se você olhar essa ferramenta (Nota do tradutor: o autor refere-se ao Adobe Flash CS4), vai perceber que ela foi criada para Designers. Existe um “stage”, a linha do tempo, ferramentas de desenho, e assim por diante.

Quando você está desenvolvendo aplicações e você se preocupa com produtividade, você precisa de componentes, precisa de ser capaz de agilizar o desenvolvimento, tanto através de reutilização de código e, não menos importante, quanto a uma IDE que dê conta do recado.

Assim, a resposta pode ser revista: Flex é um framework open source que ajuda os desenvolvedores a criar rapidamente aplicações para a Internet, que rodam pelo Flash Player. O framework possui este conceito desde 2006, com a versão Flex 2 e o Flash Player 9, e o Action Script 3. A versão atual é o Flex 3, e no início de 2010 a versão Flex 4 será lançada.



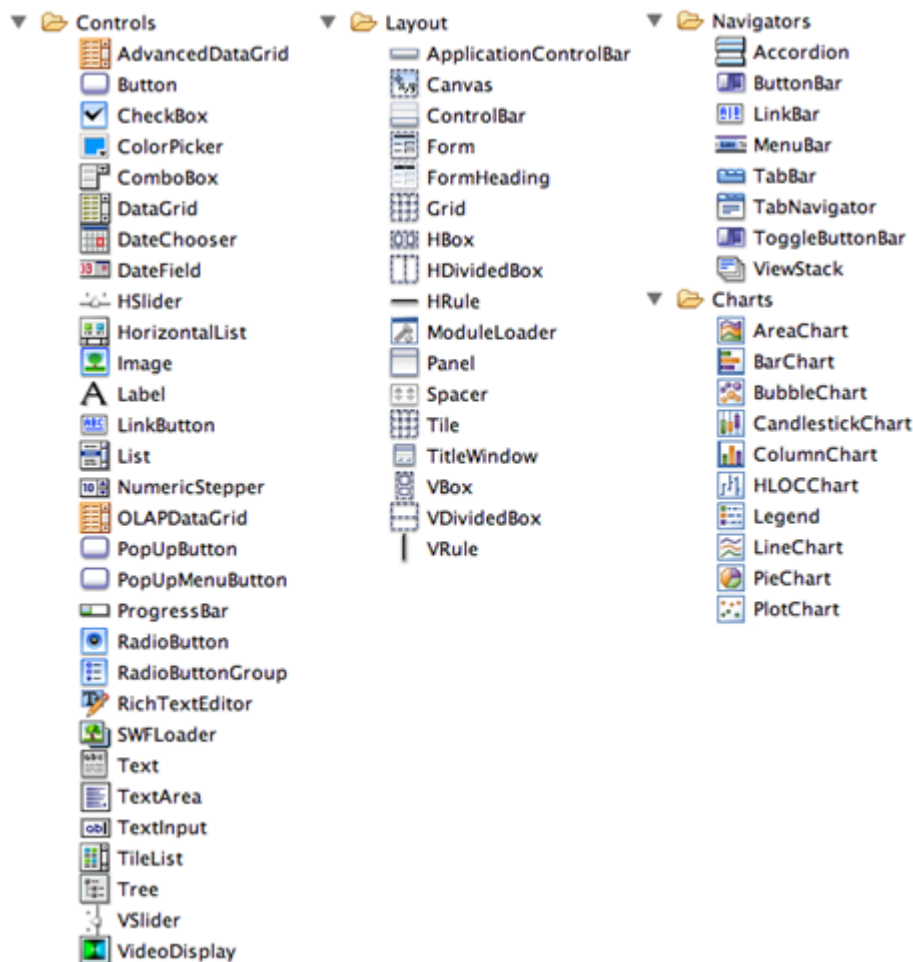
## Flex: duas linguagens, um único framework

Sob o aspecto do Flex, você irá encontrar o seguinte:

- Duas linguagens: MXML e Action Script 3. Flex oferece duas linguagens para que se possa criar uma aplicação Flex. Nos próximos capítulos, iremos estudar mais a fundo cada uma.
- Uma rica biblioteca de componentes
- Um compilador e um debugador
- Ferramentas disponíveis através de linha de comando, para compilação e debug de uma aplicação flex.

Desde que o Flex se tornou um framework *open source*, eu encorajo a todos para acessar a página do projeto em <http://opensource.adobe.com/flex> e realizar o download do SDK. Você pode acessar o código fonte de todos os componentes do framework, conferir a página de bugs (<http://bugs.adobe.com/flex>), dados sobre novas funcionalidades que são implementadas a cada versão, e a página wiki contendo especificações do framework.

Parte do ganho de produtividade fornecida pelo uso do Flex é devido a sua extensa biblioteca de componentes. Há todos os componentes de interface que o usuário imagina (*inputs, panels, Windows, sliders, data grids, combo, accordion, tab* etc). Existem *containers* e elementos de formulário. A seguir, você pode ver uma *screenshot* dos componentes visuais disponíveis do Flex 3.



E se estes componentes não são suficientes, você pode acessar o seu código fonte e estender-los para construir seus próprios componentes, ou pode criar novos componentes do zero.

## Porque você deve se preocupar com Flex

Antes de ir mais a fundo sobre o que é Flex, vamos fazer uma pausa e revisar as razões sobre o que devemos nos preocupar sobre Flex.

As aplicações web tradicionais (HTML) são formadas através de uma arquitetura *requisição-resposta*. O navegador realiza uma requisição ao servidor, que por sua vez responde com uma página de volta, e este ciclo se repete. HTML e CSS são uma excelente escolha para apresentação de informações, sem dúvida umas das melhores existentes. Com o passar dos anos, esta arquitetura se modernizou de uma exibição de conteúdo estático para uma plataforma de aplicações. Com tecnologias de script, conseguimos criar páginas dinâmicas que se adequaram a arquitetura *requisição-resposta*. Além disso, adicionando DHTML e Ajax, tivemos um novo avanço nas páginas web: o usuário poderia interagir com a página carregada e alterar partes do seu conteúdo, sem atualizar a página inteira.

Como estas tecnologias evoluíram, aplicações mais complexas apareceram. Algumas aplicações web começaram a copiar as funcionalidades das aplicações desktop e ao mesmo tempo manter as premissas de uma aplicação web (disponível em qualquer lugar e em qualquer navegador através de uma conexão com a Internet). Neste embalo, versões online de planilhas e editores de texto surgiram.

No entanto, a partir de uma perspectiva da usabilidade, as aplicações web eram menos amigáveis do que as aplicações desktop. Ao mesmo tempo, para criar estas aplicações mais complexas, precisava-se de uma série de funcionalidades de diversas tecnologias (Javascript, DHTML, CSS, Ajax, tecnologias no lado do servidor), e você precisava ter experiência com as diferenças entre os navegadores e como eles implementavam HTML/CSS/Javascript.

Assim, em 2002, a Macromedia trouxe o termo RIA (Rich Internet Applications), para descrever uma nova geração de aplicações que combinavam as vantagens das aplicações web com as vantagens das aplicações desktop. A tecnologia que tornou isso possível era o Flash Player.

Basicamente, se você desejava criar uma aplicação (Não meramente um site ou uma página Web), você poderia fazê-lo usando Flex. Algumas coisas simplesmente não são possíveis com HTML/Javascript, outras podem ser muito difíceis de implementar de forma consistente em todos os navegadores. O Flash Player oferece um dos melhores gráficos e é instalado em 98% dos computadores conectados na Internet, e trata sons e imagens como cidadãos de primeira classe. Tem suporte a microfones e webcams, suporte a *streaming* e envio de dados no formato *push*, excelente suporte a tipografia e com uma lista de qualidades sempre aumentando.

Dê uma olhada nestes três produtos para ter uma idéia do que é possível fazer usando Flex:

- [Sumopaint](#): uma aplicação para edição de imagens
- [Mindomo](#): uma aplicação de *mind mapping*
- [Times Reader](#): do New York Times

Com o passar do tempo, outras tecnologias começaram a usar o conceito RIA. Através do avanço do AJAX aplicações puderem ser feitas como o *Gmail* e o *Google Spreadsheets*, e hoje temos também o *Silverlight* da *Microsoft* e o *JavaFX* da *Sun*.

## De um cliente leve para um cliente esperto/rico

Vamos voltar aos navegadores e como eles funcionam. Quando o navegador faz uma requisição, o servidor usa a combinação de conteúdo estático (HTML / CSS / Javascript) e scripts (estes scripts podem consultar o banco de dados, chamar outros scripts, mas no final eles renderizam sempre HTML / CSS / Javascript) para criar uma página web. Esta página é carregada e processada pelo navegador. Um elemento chave neste processo é que, usualmente, a página (ou resposta) tem o código que configura a apresentação dos dados juntamente com estes dados, tudo junto.

Quando um novo estado da aplicação está para ser apresentado, o navegador faz um novo pedido e o servidor prepara a página. O cliente “apenas” renderiza a página.

Aplicações Flex funcionam de forma diferente. O servidor envia para o navegador o aplicativo Flex (o arquivo swf) que funciona dentro do navegador que possui o *plugin* do FlashPlayer. Normalmente, este arquivo SWF possui apenas a lógica de negócios que será usada no cliente. Se for necessário obter dados (de um banco de dados, por exemplo), a aplicação Flex realizara a requisição para obter estes dados. Então o servidor retorna apenas estes dados (pode ser em XML, Json ou AMF3), e o cliente sabe como representar estes dados visualmente. O que temos aqui é uma arquitetura orientada a serviços: A aplicação Flex é o cliente, um cliente que pode consumir serviços de dados do servidor. A aplicação pode mudar de estado sem atualizar a página ou recarregar o arquivo SWF no navegador. A aplicação é um cliente que pode fazer mais do que “apenas” processar dados. Assim, usando Flex e Flash Player, é possível criar praticamente qualquer coisa que faça sentido para implementação na Web, desde jogos, até aplicações e widgets que se integram nas aplicações web clássicas, e muito mais.

Mas chega de teoria, vamos ver algum código!

## Introdução à linguagem MXML

MXML é uma linguagem de marcação de texto, baseada em XML. Em uma aplicação Flex você usa MXML para rapidamente criar a estrutura/aparência de sua aplicação. No Flex, qualquer coisa que você pode fazer utilizando MXML você também pode fazer usando Action Script 3. O inverso, porém, não é verdade.

Se você pode usar o ActionScript 3 para fazer qualquer coisa que você faz com MXML, porque utilizar MXML primeiro? Geralmente é muito mais fácil acompanhar e compreender uma estrutura de interface usando uma linguagem XML (declarativa) que a linguagem ActionScript (imperativa). E isso se traduz em menos código para escrever ao criar uma interface. Também

é muito mais fácil construir ferramentas para linguagens declarativas do que linguagens imperativas. Aqui está um exemplo do “Hello World” implementado com a linguagem MXML:

```
1: <Label text = "Hello World!" fontsize = "14" color = "red" x = "100" y = "50" />
```

Nesse código, eu uso um componente Flex chamado *Label*, utilizado para exibir texto na tela da aplicação. Eu defini o atributo *text* para o texto que eu quero que seja exibido. Além disso, eu quis customizar (um pouco) a aparência e a posição do *Label* na tela. Para isso, eu uso os atributos *fontSize*, *color*, *x* e *y*.

Agora, considere o mesmo exemplo implementado usando ActionScript 3:

```
1: var myLabel = new Label();
2: myLabel.text = "Hello World!";
3: myLabel.setStyle("fontSize", "14");
4: myLabel.setStyle("color", "red");
5: myLabel.x = 100;
6: myLabel.y = 50;
7: addChild(myLabel);
```

Eu fiz sete linhas de código para fazer o que eu fiz em MXML com somente uma tag e alguns atributos. Agora, imagine que em uma aplicação real você tenha que fazer muitos controles, agrupados em diversos lugares. É muito mais fácil manter o código que está escrito em MXML do que código escrito com centenas de linhas em ActionScript.

Embora você possa usar MXML para descrever sua aplicação, você não pode usar MXML para implementar a lógica de negócios. Para isso, você deve usar ActionScript 3.

Aplicações em Flex rodam no Flash Player, e Flash Player entende apenas Action Script 2 e Action Script 3. Isto significa que qualquer código MXML que você escreve na sua aplicação deve ser transformado pelo compilador em código Action Script 3. O código então é transformado em *bytecodes* pelo compilador (o arquivo SWF), que será entendido pelo Flash Player.

Assim, quase todos os componentes MXML são classes em Action Script 3 (na verdade, existem algumas tags de MXML que não tem uma classe Action Script correspondentes, tais como *script* e *Model*). Por exemplo, aqui está um trecho da classe *Label*:

```
1: public class Label extends UIComponent
2:     implements IDataRenderer, IDropInListItemRenderer,
3:     IListItemRenderer, IFontContextComponent
4:
5: {
6:     /**
7:     * Constructor.
8:     */
9:     public function Label()
```

```
10:     {
11:         super();
12:
13:         // this is so the UITextField we contain can be read by a screen-reader
14:         tabChildren = true;
15:     }
16:
17:     /**
18:     * @private
19:     * Flag that will block default data/listData behavior.
20:     */
21:     private var textSet:Boolean;
22:
23:     ...
24: }
```

Em qualquer aplicação Flex você tem pelo menos um arquivo MXML, que é a aplicação principal. Por exemplo, aqui está o código completo para a aplicação “Hello World”:

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
3:     <mx:Label text="Hello World!" fontSize="14" color="red" x="100" y="50"/>
4: </mx:Application>
```

O nó raiz deve ser sempre *Application*, e é aqui que os *namespaces* são definidos. Neste caso eu tenho somente um *namespace* para a linguagem MXML e para os componentes Flex: *mx* (O código acima está em Flex 3, e existem poucas diferenças em relação ao Flex 4, além de mais *namespaces* para serem declarados).

Se você escreve seus próprios componentes personalizados, você deve adicionar um *namespace* à eles. Aqui, por exemplo, eu declaro um segundo *namespace* que refere-se a todos os componentes que eu criei (neste exemplo, eu usei um *Label* customizado chamado *MyCustomLabel*):

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:local="*">
3:     <mx:Label text="Hello"/>
4:     <local:MyCustomLabel text="world!"/>
5: </mx:Application>
```

Neste ponto você pode se perguntar como lidar com as diferentes páginas de uma aplicação Flex. Para um site HTML, diferentes estados são geralmente implementados em diferentes páginas. Uma aplicação Flex é muito parecida com uma aplicação *Desktop*. Isto significa que você pode usar apenas um arquivo MXML e exibir nesta página os diferentes estados da aplicação. Flex oferece uma série de maneiras de fazê-lo, a partir de componentes do Flex, como *Accordions*, navegadores em *Tabs*, layouts do tipo *Cards* e módulos.

Você viu que o código MXML pode ser usado para definir a aparência da aplicação. Mas você também pode usá-lo para criar componentes personalizados, estendendo dos componentes existentes do Flex. Vamos ver um exemplo. Suponha que você tem na sua aplicação uma série de formulários que possuem dois botões: Salvar e Cancelar. Dê uma olhada no código para este componente MXML personalizado (o código é criado dentro do arquivo `FormButtons.mxml`; todos os arquivos MXML devem ter a extensão `mxml`):

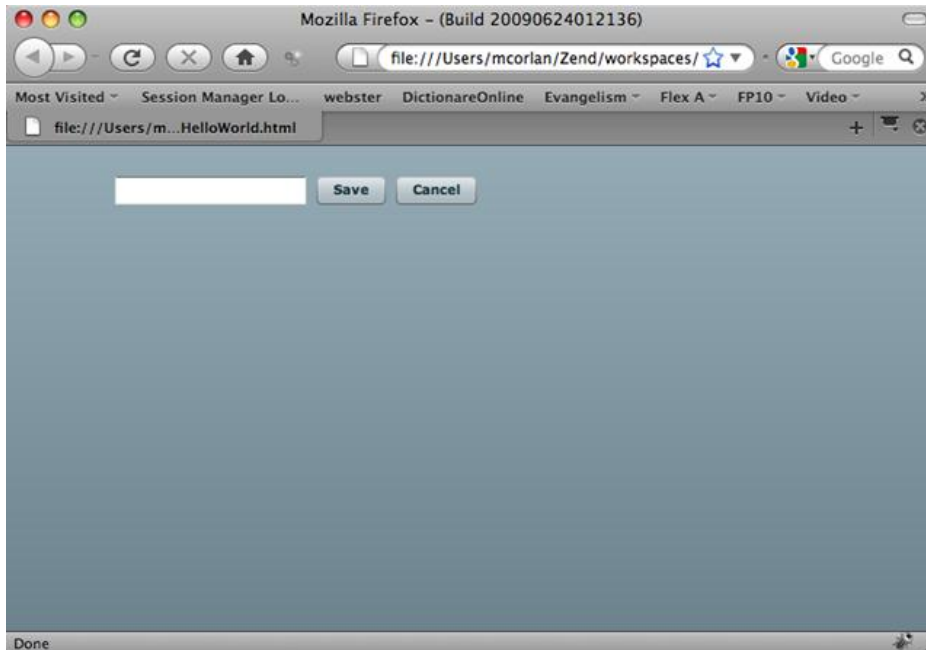
```
1: <? Xml version = "1.0" encoding = "UTF-8"?
2: <mx:HBox xmlns:mx= "http://www.adobe.com/2006/mxml" width= "400" height= "150">
3: <mx:Button id= "saveButton" label= "Save" />
4: <mx:Button id= "cancelButton" label= "Cancel" />
5: </ mx: HBox>
```

Quando você cria um componente personalizado, você pode escolher qual componente irá estender (exceto *Application*). Eu escolhi para estender *HBox* (Caixa horizontal), que é um componente que exibe todos os seus filhos dispostos horizontalmente, na mesma linha. Dentro do componente, eu adicionei dois botões, um para salvar e outro para cancelar. Eu também defini o atributo `id` para cada um. Você usa o valor de `id` como uma forma de fazer referência ao objeto em outras partes do código. É a mesma coisa que declarar uma variável em Action Script 3.

Agora vamos ver como você poderia usar este componente personalizado em uma aplicação Flex:

```
1: <? Xml version = "1.0" encoding = "UTF-8"?
2: <mx:Application xmlns:mx= "http://www.adobe.com/2006/mxml" xmlns:local= "*" layout=
"horizontal">
3: <mx:TextInput width= "150" />
4: <local:FormButtons/>
5: </ mx: Application>
```

A seguir você pode verificar como a aplicação ficou:



Talvez você possa pensar que em MXML você pode usar apenas componentes visuais (isto é, os componentes que acabam sendo exibidos). Mas isso não é verdade. Você pode ter tags MXML que representam dados (objetos que são utilizados para armazenar dados) ou de componentes que manipulam dados (componentes que podem recuperar ou enviar dados de ou para um servidor). Abaixo você pode ver um exemplo do componente *Object*, que tem a propriedade *name*.

```
1: <mx:Object id= "myObj" name= "Mihai Corlan" />
```

Como eu disse antes, tudo (bem, quase tudo) que vem do Flex é representado por uma classe ActionScript que implementa uma aparência visual ou de lógica. Quando você escolher criar um componente personalizado (componente visual ou não) usando ActionScript, você tem que ter em mente que existe uma restrição: o construtor desta classe deve ser capaz de ser chamado sem argumentos (e se existem argumentos, eles devem ter valores padrão).

## Misturando MXML e ActionScript 3

Voltando ao componente personalizado *FormButtons* (aquele com dois botões), você pode ter encontrado um problema: E se você quiser usar esse componente em um lugar onde os textos “salvar” e o “cancelar” não fazem sentido. Você precisa criar um outro componente personalizado com os textos que deseja (por exemplo “mostrar” e “ocultar”)? Claro que essa é uma opção, mas não é escalável ou elegante. O que você realmente quer é um componente mais genérico, e uma maneira para alterar o componente usando código. É por isso que, mais cedo ou mais tarde, você terá que escrever código ActionScript, para ir além do MXML.

No próximo exemplo, eu adicionei o código ActionScript dentro do código MXML do componente para definir duas variáveis que armazenam os textos dos botões utilizados. Veja que estou utilizando uma nova tag, *Script*, e dentro desta, CDATA. Isso ocorre porque dentro de documentos XML caracteres como >, < & são ilegais se não houver um tratamento prévio (escape). Aliás, eu não comentarei muito sobre o código ActionScript agora, deixarei para ir mais a fundo nas seções posteriores.

```

1: <? Xml version = "1.0" encoding = "UTF-8"?
2: <mx:HBox xmlns:mx= "http://www.adobe.com/2006/mxml" width= "400" height= "150">
3: <mx:Script>
4: <! [CDATA [
5: public var label1: String = "Salvar";
6: public var label2: String = "Delete";
7: ]]>
8: </ mx: Script>
9: <mx:Button id= "saveButton" label= "{label1}" />
10: <mx:Button id= "cancelButton" label= "{label2}" />
11: </ mx: HBox>

```

As variáveis definidas podem ter seus valores alterados a partir da aplicação Flex que usa este componente. Então vamos ver como o código da aplicação Flex fica ao usar este novo compoennte personalizado:

```

1: <? Xml version = "1.0" encoding = "UTF-8"?
2: <mx:Application xmlns:mx= "http://www.adobe.com/2006/mxml" xmlns:local= "*" layout=
"horizontal">
3: <mx:TextInput width= "150" />
4: <local:FormButtons label1= "Show" label2= "Hide" />
5: </ mx: Application>

```

Observe que a tag *FormButtons* possui dois atributos: *label1* e *label2*. Assim podemos definir o texto que você deseja exibir nos botões. E este é o mecanismo que você usa para adicionar mais funcionalidades a um componente MXML (usando código ActionScript). Em um aplicativo real, você pode querer vincular um comportamento para cada botão, para que quando ele for clicado, algo aconteça. Você usa código ActionScript para escrever as funções que serão desencadeadas ao pressionar os botões.

Há um segundo método para adicionar código ActionScript no MXML. Você pode criar um arquivo ActionScript (neste caso é *buttons.as*), e incluir esse arquivo no arquivo MXML. Você faz isso adicionando a tag *Source*, utilizando o atributo *script* que aponta para o arquivo AtionScript. Aqui está o código desta abordagem:

```

1: buttons.as / ActionScript arquivo chamado
2: public var label1: String = "Salvar";
3: public var label2: String = "Delete";

1: <? Xml version = "1.0" encoding = "UTF-8"?
2: <! Componente MXML - FormButtons.mxml um arquivo chamado ->
3: <mx:HBox xmlns:mx= "http://www.adobe.com/2006/mxml" width= "400" height= "150">
4: <mx:Script Source= "buttons.as" />

```

```
5: <mx:Button id= "saveButton" label= "{label1}" />
6: <mx:Button id= "cancelButton" label= "{label2}" />
7: </ mx: HBox>
```

Agora vamos voltar um pouco e entender o que está acontecendo quando o compilador MXML analisa o arquivo *FormButtons.mxml*. Você já sabe que todo código será transformado em código *ActionScript*. Mas o que acontece com o código *ActionScript* existente que eu adicionei (as duas variáveis)? O compilador compila cada arquivo MXML em uma classe *ActionScript*. Neste caso eu vou ter uma classe chamada *FormButtons* (porque este é o nome do arquivo e é usado para o nome da classe) que estende de *HBox* (porque eu escolhi *HBox* como o nó raiz do componente). E todo o código *ActionScript* dentro da classe torna-se membro da classe: variáveis (como os do exemplo) tornam-se variáveis de instância, e as funções tornam-se métodos de instância.

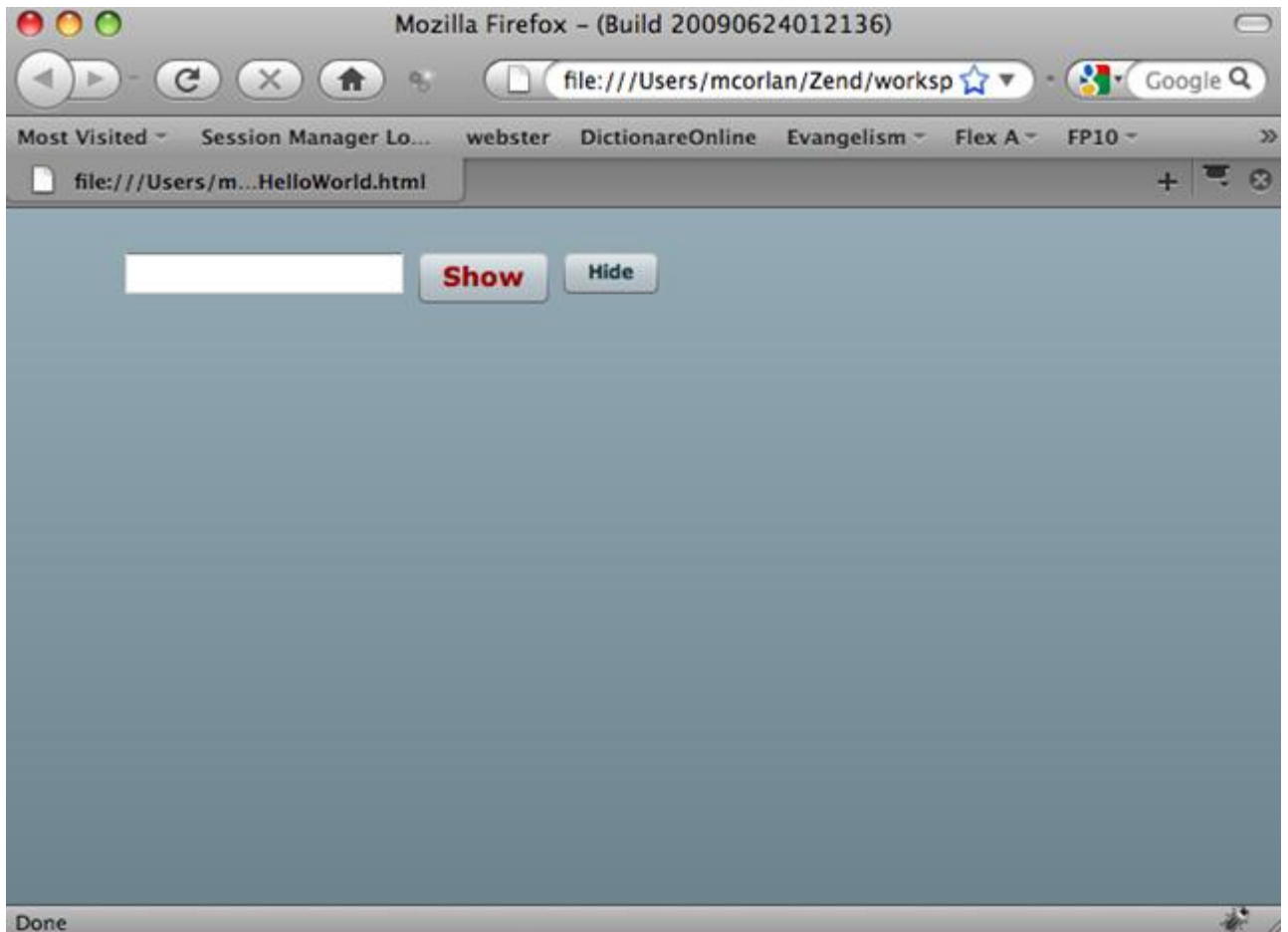
## Estilos CSS

Neste ponto você deve estar se perguntando se é possível alterar a aparência visual dos componentes do Flex. Existe algo parecido com CSS para HTML? A resposta é sim, Flex suporta CSS. Em Flex 4, o uso de CSS é estendido para permitir a definição de estilos, não só com base em nome de classe, mas em identificações, para permitir pseudo-seletores (por exemplo, para um botão que você tem o seletor para baixo, sobre o seletor, e assim por diante) e muito mais.

Tal como em HTML, podem ser definidos estilos *inline* (dentro do próprio código MXML) ou em um arquivo separado. Vamos voltar para o componente personalizado, *FormButtons*, e definir alguns estilos. Se você optar por definir os estilos em um arquivo separado, você usa a tag *style* e define o caminho para o arquivo de estilo dentro do atributo de origem.

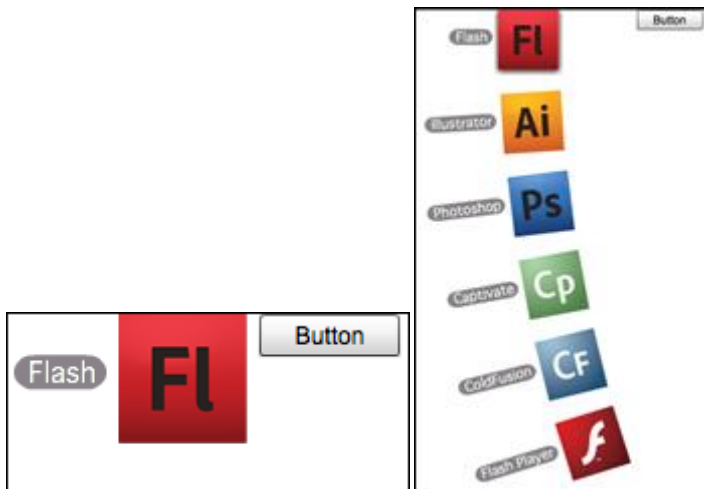
```
1: <? Xml version = "1.0" encoding = "UTF-8"?
2: <! Componente MXML - FormButtons.mxml um arquivo chamado ->
3: <mx:HBox xmlns:mx= "http://www.adobe.com/2006/mxml" width= "400" height= "150">
4: <mx:Style>
5: . Button1 (
6: font-size: 14;
7: color: # 990000;
8:)
9: </ mx: Style>
10: <mx:Script Source= "buttons.as" />
11: <mx:Button id= "saveButton" styleName= "Button1" label= "{label1}" />
12: <mx:Button id= "cancelButton" label= "{label2}" />
13: </ mx: HBox>
```

Eu criei uma classe chamada *Button1* que define a cor do texto e do tamanho da fonte. Então defini este estilo para o primeiro botão usando o atributo *styleName* do primeiro botão. A aplicação agora parece com isto:



Estilos CSS podem ser alterados em tempo de execução (após a aplicação Flex ser carregada no navegador), e a aparência do aplicativo irá mudar imediatamente.

No Flex 4, Adobe adicionou uma nova linguagem chamada "MXML para gráficos" que adiciona gráficos primitivos, efeitos, máscaras e transformações em 2D. Você pode criar uma classe *skin* em um arquivo MXML utilizando estas novas tags, e então você pode definir essa classe para o componente que você deseja personalizar. Abaixo você pode ver uma lista de skins no Flex. A imagem à esquerda mostra a lista em seu estado padrão, e a da direita mostra o estado quando o mouse passa sobre o componente (chamado de *hover*). Você pode ver a aplicação [aqui](#).



Enquanto estuda a documentação de aparências para as aplicações Flex, você pode pesquisar sobre o termo “skinning”. Você pode fazer uma *skin* gráfica ou programática, a fim de mudar a aparência. [Aqui](#) está um bom artigo sobre isso.

## Modificando o código MXML em tempo de execução

Às vezes você quer mudar os componentes de interface em tempo de execução. Talvez você queira, dependendo de alguns dados que obtiver do servidor, construir um formulário dinâmico. Novamente, você pode escrever código ActionScript para fazer isso. Qualquer componente visual em Flex tem métodos para adicionar um novo componente, ou para removê-lo, obter todos os componentes, e assim por diante. Se quiser, você pode comparar isto com o mesmo modo de alterar a DOM do HTML usando JavaScript. No entanto, há uma diferença: com JavaScript você pode injetar código HTML dinamicamente, que veio do servidor (com uma chamada Ajax). No Flex, isso não é possível, e a função *eval()* não existe. No entanto, existe outra maneira de carregar outras aplicações ou módulos, após a aplicação principal ser carregada.

Quando você sabe todos os estados possíveis de um componente, você pode usar *States* para implementar diferentes visualizações para o mesmo componente ou aplicativo. *States* é um conceito que melhorou substancialmente no Flex 4 em relação do Flex 3, tornando tudo muito mais fácil e poderoso. Em Flex 4, você trabalha com *States* da seguinte forma:

- Você define um número de *States* e escolhe o padrão
- Então você especifica como e quando cada componente deve aparecer no *State*
- Você pode especificar valores diferentes para qualquer atributo e para qualquer estado onde ele aparece

Suponha que você queria criar um componente que lida com as funcionalidades de *login* de uma aplicação. Você quer usar esse componente para exibir o formulário de *login*, e quando o *login* for bem sucedido, você deseja exibir o botão sair e o nome do usuário.

A seguir o código para mostrar como criar esse componente de Login/Logout usando Flex 4:

```
<?xml version="1.0" encoding="utf-8"?>
  2: <s:Group xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark" xmlns:mx="library://ns.adobe.com/flex/halo">
  3:   <s:states>
  4:     <s:State name="notlogged"/>
  5:     <s:State name="logged"/>
  6:   </s:states>
  7:
  8:   <s:TextInput includeIn="notlogged" text="user name"/>
  9:   <s:TextInput includeIn="notlogged" text="password"/>
 10:   <s:Button includeIn="notlogged" label="Login" click="{currentState='logged'}/>
 11:
 12:   <mx:Label includeIn="logged" text="Current user: Mihai Corlan"/>
 13:   <s:Button includeIn="logged" label="Logout" click="{currentState='notlogged'}/>
 14:   <s:layout>
 15:     <s:HorizontalLayout/>
 16:   </s:layout>
 17: </s:Group>
```

O código se explica por si só. Como componente principal, eu usei o componente *Group*, que tem a *layout* setado para *HorizontalLayout* (possui o mesmo efeito que o *HBox* do Flex 3). Eu defini os *States* disponíveis para este componente no início do código. Depois, criei os botões, caixas de texto e *labels*. O atributo *includeIn* especifica em qual *State* o componente irá aparecer. Somente para registrar, existe também o atributo *excludeFrom*. Se você precisa deixar um botão disponível em todos os *States*, basta não especificar nenhum destes atributos. Finalmente, você pode ver que eu atribuí uma expressão ao atributo *click* dos dois botões. Por exemplo, *click="{currentState='logged'}*" diz ao Flex que quando o botão for clicado, o estado (*State*) do componente será alterado para *logged*.

Incluí mais código *ActionScript* neste exemplo, embora eu ainda esteja falando sobre a linguagem *MXML*. Este é um sinal que é hora de passar para a segunda linguagem do Flex, o *ActionScript 3*.

## Introdução à linguagem Action Script 3

*Action Script 3* é uma linguagem de script orientada a objetos, que é (na maioria das vezes) segura. *ActionScript 3* é baseado na especificação *ECMAScript 3* (EMCA-262). Além disso, algumas características suas são alinhados com a proposta do *ECMAScript 4*. Descobri que é mais fácil explicar o *ActionScript 3* para alguém que não conhece a linguagem, da seguinte forma: *ActionScript* parece uma mistura de *JavaScript* e *Java* além de ter a sua própria

identidade. Na verdade, o Javascript é uma outra linguagem baseada nas especificações do ECMAScript, assim, é natural que têm coisas em comum com o ActionScript.

Como eu disse antes, o Flash Player pode trabalhar com duas linguagens: ActionScript 2 e ActionScript 3. Internamente, ela usa duas máquinas virtuais diferentes, para trabalhar com estas duas linguagens (ActionScript3 e AVM2 Virtual Machine surgiu no Flash Player 9). ActionScript 3 é constituída de um núcleo (tipos, constantes, e assim por diante) e da API do Flash Player (esta API fornece aos desenvolvedores o acesso a todas as funcionalidades do Flash Player através da lista de exibição API, 3D API, API de desenho, animações, e assim por diante). Neste artigo, vou me concentrar nesta linguagem. [Aqui](#) está um bom artigo introdutório sobre o ActionScript3.

De agora em diante vou usar a abreviatura “AS3” ao invés de “ActionScript 3”.

## Separando as Declarações

No PHP você usa ponto e vírgula (;) para separar ou terminar uma declaração. Em AS3 pode utilizar um ponto e vírgula também, ou simplesmente o final da linha. Devo dizer que quando eu vejo o código escrito sem o ponto e vírgula, não é uma alegria para os meus olhos. Então eu sugiro que você use o mesmo método que usa no PHP.

## Os tipos de dados, variáveis e constantes

No PHP existem os seguintes tipos de variáveis: booleanos, inteiros, ponto flutuante, strings, arrays, objetos, resources e NULL.

Em AS3 temos:

- Tipos nativos: Boolean, int, uint (mesmo que o ponto flutuante), Strings, NULL (contém apenas um valor: null), void (contém apenas um valor: indefinido)
- Tipos complexos: Object, Array, Vector (criado no Flash Player 10), Dicionário, Bitmap, ByteArray, Data, XMLList, Function, Erro, RegExp.

Em AS3, uma variável é apenas um identificador ou uma referência associada ao valor real. Os valores reais em AS3 são *object* (int ou uint são objetos, o mesmo para *Number* ou *Date*), *null* e *undefined*. *Null* e *undefined* significam ausência de dados, no entanto, há uma diferença entre eles. Quando você declarar uma variável e não inicializá-la, ela terá o valor *null* se o tipo da variável não é *Boolean*, *int*, *uint* ou *Number*. Se a variável é fortemente tipada e não está inicializada, ela terá o valor *undefined*. As mesmo tempo, quando você tem um objeto dinâmico e quiser verificar se um determinado método ou propriedade é definida, pode verificar se é *undefined*.

No PHP você declara uma variável assim:

```

1: $anInteger = 12;
2: $isTrue = true;
3: $aString = "my string";
4: //or
5: $aString = 'my string';

```

Em AS3 você pode usar a palavra chave *var* ao declarar uma variável.

```

1: var anInteger:int = 12;
2: var isTrue:Boolean = true;
3: var aString:String = "my string";
4: //Em AS3 você não pode usar aspas simples para declarar strings

```

Note que após o nome da variável, eu tenho um tipo da variável, por exemplo:

*myVarName:Boolean* (o tipo é declarado usando ":"). AS3 permite trabalhar com qualquer tipo com ou sem a sua delcaração (se o compilador estiver setado para "strict mode", então você deverá informar o tipo da variável).

Vindo de PHP onde você não tem que declarar o tipo da variável, isto pode parecer estranho, e você pode ter o mesmo pensamento, ficar sem declarar o tipo da variável. Por mais tentador que isso possa parecer, recomendamos sempre que crie variáveis com seu tipo declarado. Inicialmente, quando você usa uma IDE para escrever código, definido o tipo das variáveis, permite que sejam encontrados mais erros em tempo de compilação. Por exemplo, considere uma função que tem um único argumento do tipo *String*. Se você tentar chamar esta função passando um objeto como argumento, a IDE irá alertá-lo sobre este erro.

Sem atribuir um tipo às variáveis, você irá obter erros também, mas somente quando você ou o usuário final executar a aplicação, e neste ponto o *bug* será muito difícil de encontrar.

A segunda razão para usar tipos é que o compilador AS3 pode fazer otimizações caso ele saiba dos tipos específicos das variáveis.

No PHP, você pode alterar o tipo de uma variável da seguinte forma:

```

1: $myVar = 12; //it is an int
2: $myVar = "now is a string";

```

Em AS3 você pode fazer o mesmo (em *strict mode*) somente se declarar uma variável do tipo *"\*"*:

```

1: var myVar:int = 12;
2: //isto irá gerar um erro a aplicação não compilará
3: myVar = "this is a string";
4:
5: //declarando uma variável sem tipo, você pode alterar o seu tipo quando quiser
6: var myVar2:* = 12;
7: myVar2 = "this is a string now";

```

Perceba que eu uso a palavra chave *var* somente quando declaro a variável. Para mais atribuições deve-se omitir o *var* e o tipo da variável.

Como eu disse antes, as variáveis AS3 são apenas referências para o objeto real. No entanto, quando você atribuir um *int*, *uint*, *Number*, *Boolean*, ou uma variável *String* para outra variável, será criada uma cópia (o mesmo acontece quando se passa uma variável deste tipo para uma função). No PHP, você pode usar o operador “&” para atribuir uma variável por referência, mesmo para tipos primitivos, e quando você mudar o valor para uma variável, a outra variável irá apontar para o mesmo valor alterado.

Para concatenar *strings* em PHP, você usa “.” (ponto), em AS3 você usa “+” (mais):

```
1: //in PHP
2: $space = " ";
3: $a = "this" . $space . "is!";
4:
5: //in AS3
6: var space:String = " ";
7: var a:String = "this" + space + "is!";
```

No PHP você pode definir variáveis onde quiser: no nível de arquivo, em uma função, em uma classe. Em aplicações Flex, variáveis podem ser declaradas somente dentro de uma função ou no nível de classe.

Além disso, em PHP você pode ter uma parte em programação procedural, que não é declarada dentro de uma função:

```
1: <?php
2:
3: $a = 1;
4: for ($i=0; $i<100; $i++) {
5:     $a += $i * $a;
6: }
7:
8: ?>
```

Em AS3 não pode-se fazer operações com variáveis fora de funções (embora você possa declarar variáveis fora das funções), com uma exceção que explicarei quando for apresentar Classes. Assim, se você tentar executar o código seguinte, você terá um erro quando compilar a aplicação:

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
3:     <mx:Script>
4:         <![CDATA[
5:             var a:int = 1;
6:             for (var i:int = 0; i<100; i++) { //this raises an error
7:                 a += i * a;
8:             }
9:         ]]>
10:     </mx:Script>
```

```
11: </mx:Application>
```

Você pode reescrever este código para que ele funcione:]

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
3:   <mx:Script>
4:     <![CDATA[
5:       var a:int = 1;
6:
7:       function calculations(a:int):int {
8:         for (var i:int = 0; i<100; i++) {
9:           a += i * a;
10:        }
11:        return a;
12:      }
13:    ]]>
```

Em PHP, constantes são declaradas assim:

```
1: //constants
2: define("CONSTANT", "Hello");
3: $myString = CONSTANT . ' world!';
```

No AS3 constantes são declaradas usando a palavra “const” (por convenção o nome das constantes estão em CAIXA ALTA):

```
1: static const HELLO:String = "Hello";
2: var myString:String = HELLO + " world!";
```

O que pode ser usado como nome de uma variável? Aqui PHP e AS3 são semelhantes: enquanto o primeiro caractere do nome é uma letra ou “\_”, seguido de letras, dígitos ou sublinhados. Aqui estão alguns exemplos de nomes de variáveis que são legais em ambas as linguagens: `_1`, `_a1A`, `b`.

No PHP você tem uma maneira de usar a variável conhecendo o seu nome:

```
1: <?php
2: $myVar = 12;
3: $varName = 'myVar';
4: echo($$varName); //imprime 12;
5: ?>
```

Você pode usar uma funcionalidade semelhante em AS3 utilizando o método dinâmico de referência à membros (variáveis / métodos):

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
creationComplete="init()">
3:   <mx:Script>
4:     <![CDATA[
5:
6:         var myVar:int = 12;
7:
8:         function init():void {
9:             var varName:String = "myVar";
10:            trace(this[varName]); //output 12
11:        }
12:    ]]>
13:   </mx:Script>
14:
15: </mx:Application>
```

Neste exemplo eu usei o “*this*” para referenciar o objeto atual, mas você poderia usar a mesma técnica em qualquer outro objeto. Eu irei adicionar mais exemplos quando comentar a respeito de classes dinâmicas.

## Funções e funções anônimas

Em AS3 você pode fazer tudo o que é feito com funções em relação ao PHP, e algumas coisas a mais.

Primeiro, em AS3, você pode definir o tipo de argumento e o tipo de retorno (em PHP, você pode adicionar tipos somente nos métodos dos objetos).

```
1: function calculate(x:int=0, y:int=0):int {
2:     return x + y;
3: }
4: //using the function
5: var result:int = calculate(1, 2);
6:
7: function calculateAverage(...arguments):Number {
8:     var result:Number = 0;
9:     for (var i:int=0; i<arguments.length; i++) {
10:         result += arguments[i];
11:     }
12:     return result/arguments.length;
13: }
14:
15: //using the function
16: var result:Number = calculateAverage(1, 2, 3, 4, 5, 6);
```

Você pode misturar o operador ... (chamado de *rest*) com argumentos explícitos, se e somente se colocar o operador *rest* como o último da lista de argumentos: `function foo(x:int, y:int, ... arguments):Number {}`. O operador *rest* (...) pode ser útil quando você precisa criar funções com um número variável de argumentos.

Se a função não retorna nada, você pode usar o tipo “void” como retorno.

Em PHP e AS3 você pode ter valores padrão para os argumentos. Por exemplo:

```
1: //php code
2:
3: function doThing($a, $b="default value") {
4:     echo $a . $b;
5: }
6:
7: //AS code
8: function doThing(a:String, b:String="default value"):void {
9:     trace(a + b);
10: }
```

Claro que você pode definir funções dentro de funções (você verá um exemplo no próximo exemplo).

Em segundo lugar, qualquer função AS3 é representada pela instância de uma classe chamada *Function*. Isso faz algumas coisas interessantes serem possíveis:

- Você pode criar uma função literal e atribuí-la a uma variável e, em seguida, chamar a função através dessa variável (isto é possível em PHP também)
- Você pode retornar uma função como o resultado da execução de uma outra função
- Você pode passar funções como argumentos ao chamar outras funções

```
1: var f:Function = function multiply(x:int, y:int):int {
2:     return x*y;
3: }
4:
5: trace(f(3,5));
6:
7: function giveMeAFunction():Function {
8:     return function multiply(x:int, y:int):int {
9:         return x*y;
10:    };
11: }
12:
13: var h:Function = giveMeAFunction();
14: trace(h(3,4));
```

Em PHP e AS3 você pode criar funções anônimas (ou “closures”). No código anterior, você pode ver um exemplo de como criar uma função anônima dentro do *giveMeAFunction()* e retorná-la.

Talvez a grande diferença entre as funções no AS3 e PHP seja em como defini-las. No PHP, você pode definir qualquer quantidade de funções em um arquivo. Em AS3 você pode definir somente uma função em cada arquivo, e o nome da função deve ser o mesmo do arquivo. Por exemplo, se você criar uma função chamada *doSomeMath()*, você deve criar um arquivo chamado *doSomeMath.as*. Ao definir funções você pode usar packages (você irá ver packages na próxima sessão). Assim, quando você precisar criar um grupo de funções utilitárias, se você não quiser criar um grupo de arquivos, você pode criar uma classe única e defini-los como métodos estáticos.

## OO: Classes e interfaces

É hora de passar para as características da programação orientada a objetos, tanto no PHP quanto no AS3. No PHP, você pode escrever código orientado a objetos ou procedural; no AS3 pode apenas orientado a objetos.

Vamos começar com uma classe PHP simples para ver as diferenças de sintaxe (lembre-se que eu uso PHP 5.3 como uma referência)

```
namespace org\corlan {
    2:
    3:     class SimpleClass {
    4:
    5:         public $public = 'Public';
    6:         protected $protected = 'Protected';
    7:         private $private = 'Private';
    8:
    9:         function SimpleClass() {
    10:
    11:         }
    12:         // Redefine the parent method
    13:         function displayVar()
    14:         {
    15:
    16:         }
    17:     }
    18: }
    19:
    20: //use the class like this
    21: require_once('flassFile.php');
    22: $obj = new org\corlan\SimpleClass();
    23: $obj->displayVar();
```

No AS3 a mesma classe é escrita da seguinte forma:

```
1: package org.corlan {
2:
3:     public class SimpleClass {
4:
5:         public var _public:String = "Public";
6:         protected var _protected:String = "Protected";
7:         private var _private:String = "Private";
8:
9:         function SimpleClass() {
10:
11:         }
12:
13:         // Redefine the parent method
14:         public function displayVar():void
15:         {
16:
17:         }
18:     }
19:
20: }
21:
22: //you use the class like this:
23: import org.corlan.SimpleClass;
24:
25: var object:SimpleClass = new SimpleClass();
26: object.displayVar();
```

Aqui estão as diferenças principais:

- Nome do arquivo onde a classe é armazenada
  - No PHP você pode definir uma classe em um arquivo que pode ser chamado de qualquer maneira
  - Em AS3, o nome do arquivo deve ser o mesmo que o nome da classe (se a classe é chamada *SimpleClass*, em seguida, o arquivo deve ser *SimpleClass.as*)
- Packages VS Namespaces
  - Em PHP você pode usar *Namespaces*, para evitar colisões no nome das classes.
  - Em AS3, você pode usar *Packages*, mas quando criar uma classe que está no *Package org.corlan*, significa que a classe está dentro da pasta *org / corlan* a partir da classe de origem do Flex. O *Package* se traduz em uma estrutura de pastas. *Packages* são utilizados em conjunto com visibilidade de classes, que podem esconder determinadas classes que estão fora do projeto (mais sobre isso mais tarde).
- require/include VS import

- No PHP, normalmente, você inclui o arquivo onde a classe é definida usando a função *require\_once*. A partir do PHP 5, você pode definir a função *\_\_autoload()* e fazer o *require\_once* ou *include\_once* nesta função em vez de escrever uma lista de arquivos a serem incluídos para cada arquivo.
- Em AS3, você usa o comando *import* para incluir a classe desejada. No entanto, se você quiser incluir todas as classes do *Package org.corlan*, você pode usar o comando *import* usando asterisco: *import org.corlan.\**. Outra diferença é que o compilador AS3 irá somente compilar as classes que são realmente utilizados em seu código (ou seja, quando a instância é criada).
- Chamar um método / membro de uma instância:
  - No PHP, você usa o operador “->”
  - No AS3, você usa o operador ponto: “.”

Agora, vamos ver os modificadores de classe, métodos e membros.

Vou começar com modificadores de classe:

- PHP tem *final* e *abstract*; *public* está implícito, todas as classes são públicas em PHP
- AS3 tem *public*, *internal*, *final* e *dynamic*. Se você não especificar qualquer modificador de acesso (*public* ou *internal*) então a classe é, por padrão, *internal*, o que significa que só podem ser acessados pelas classes do mesmo *Package*. *Public* e *final* tem o mesmo significado em PHP; *abstract* não existe no AS3, mas você pode contornar esta limitação usando interfaces. *Dynamic* marca a classe como uma classe que pode ser alterada em tempo de execução, modificando os membros existentes ou adicionando novos.

Modificadores para propriedades de classes:

- PHP: *public*, *private*, *protected*, *static*
- AS3 tem os mesmos modificadores de PHP e mais o *internal*. *Internal* é usado para fazer propriedades disponíveis somente dentro do mesmo pacote. Quando nenhum modificador for especificado, o *internal* é usado.

Modificadores para métodos das classes:

- PHP tem *public*, *private*, *protected*, *static*, *final* e *abstract*
- AS3 tem: *public*, *private*, *static*, *final*, *internal* e *override*. *Abstract* não existe no AS3; *Internal* deixa os métodos disponíveis somente nas classes que estão no mesmo *Package*.

Construtores em PHP podem ser marcados como *private*, você pode definir um construtor com o mesmo nome da classe ou você pode usar os métodos especiais `__construct()` e `__destruct()`. O construtor do AS3 é sempre público e deve ter o mesmo nome do nome da classe. Se nada for fornecido, o compilador irá criar um construtor por baixo dos panos.

Membros estáticos ou métodos estáticos são acessados:

- Em PHP usando `ClassName::propertyName`
- Em AS3, usando `ClassName.propertyName`. No entanto, dentro da mesma classe, você não precisa informar o nome da classe.

```
1: package org.corlan {
2:
3:     public class Foo {
4:
5:         private static myVar:String;
6:
7:         function Foo() {
8:             Foo.myVar = "value 1";
9:             myVar = "value 2";
10:        }
11:
12:    }
13: }
```

Uso do “this”

- Em PHP você usa a variável de classe especial `$this` para referenciar os membros da classe (variáveis/métodos) definidos na mesma classe: `$this->myVar = 22;`
- Em AS3 você usa o mesmo `this`: `this.myVar = 22;` entretanto, você pode tirar o `this` e usar somente `myVar=22;`

Em AS3 somente uma classe pode ser declarada dentro de um *Package* (A classe dá o nome do arquivo). No entanto, fora da declaração *Package*, você pode criar quantas classes quiser.

```
package org.corlan {
2:
3:     public class Foo {
4:
5:         private static var instance:Foo;
6:
7:         function Foo(object:Bar) {
8:
9:         }
10:
11:         static public getInstance():Foo {
```

```
12:         if (Foo.instance == null) {
13:             Foo.instance = new Foo(new Bar());
14:         }
15:         return Foo.instance;
16:     }
17:
18: }
19:
20: class Bar {}
```

Isto tem um efeito interessante: todas as classes definidas em um arquivo `for a` do pacote estarão disponíveis apenas para o código declarado dentro do mesmo arquivo. Para todos os outros códigos, a classe não existe. Lembra da limitação em AS3 de não poder criar um construtor privado. Bem, usando uma técnica similar a este exemplo, você pode ter certeza que existe apenas uma instância da classe `Foo`. Se algum código externo chama o construtor, uma exceção será executada, porque o código de fora não pode usar uma instância de `Bar` como esta classe.

## Herança

Estender as classes em AS3 é semelhante ao que é feito em PHP. Você utiliza a mesma palavra chave *extends* seguida do nome da classe que você deseja estender. Fazer *override* é como no PHP, a única diferença é que você tem que adicionar a palavra chave *override* antes da assinatura do método. Sobrecarga não é suportado (você não pode ter dois ou mais métodos com o mesmo nome).

No PHP você acessa os membros da classe pai com a seguinte sintaxe: *memberName*; em AS3 você usa: *super.memberName*. Quando o construtor de classe é executado, inicialmente o construtor da classe pai é chamado. Isso acontece mesmo quando você não o chama explicitamente no código. Dessa forma, você dá uma chance para a classe pai ser inicializada corretamente, para que a classe filha não use os membros que ainda não foram definidos. Você chama o construtor da classe pai usando a sintaxe *super()*. Vamos ver esses conceitos em ação, primeiro o código PHP e, em seguida, o código AS3.

```
1: class SimpleClass {
2:
3:     function SimpleClass() {
4:         echo('SimpleClass() called');
5:     }
6:
7:     function __construct() {
8:
9:     }
10:
11:
12:     function displayVar()
13:     {
```

```

14:     echo "SimpleClass class\n";
15: }
16: }
17:
18: class ExtendClass extends SimpleClass {
19:
20:     function ExtendClass() {
21:         $myVar = 1;
22:         parent::SimpleClass();
23:         //or
24:         parent::__construct();
25:     }
26:     // Redefine the parent method
27:     function displayVar()
28:     {
29:         echo "Extending class\n";
30:         parent::displayVar();
31:     }
32: }

```

```

public class SimpleClass {
2:
3:     function SimpleClass() {
4:         trace("SimpleClass() called");
5:     }
6:
7:     public function displayVar():void
8:     {
9:         trace("SimpleClass class");
10:    }
11: }
12:
13: public class ExtendClass extends SimpleClass {
14:
15:     function ExtendClass() {
16:         super(); //we have to call first the parent constructor, and only after we can
execute our code
17:         var myVar:int = 1;
18:     }
19:
20:     override public function displayVar():void {
21:         trace("overrided displayVar()");
22:         super.displayVar();
23:     }
24: }

```

Vamos dar uma olhada em como a classe é inicializada em AS3. Quando uma classe é instanciada, primeiro todas as propriedades são inicializadas, então o código estático é definido no nível da classe e é executado (isto não é possível no PHP) e, em seguida, o construtor é executado. Aqui está um exemplo:

```

1: public class Foo {

```

```

2:
3:     private var a:int = 0;
4:     private static var os:String;
5:     trace("initializer");
6:
7:     if (Capabilities.os == "LINUX")
8:         os = "LINUX";
9:     else
10:        os = "other";
11:
12:    public function Foo(a:int=1) {
13:        trace("foo() executed");
14:    }
15: }
16:
17: var foo1:Foo = new Foo();
18: var foo2:Foo = new Foo();
19: //produces this output in console:
20: initializer
21: foo() executed
22: foo() executed

```

Em AS3 você pode criar objetos fora do final da função usando o protótipo da função (isto é semelhante ao que você usa em JavaScript para criar/estender uma classe). Aqui está um pequeno exemplo:

```

1: //we create a function
2: function MyClass(value:String = "Mihai") {
3:     //we create a property
4:     this.name = value;
5: }
6: //we use the special variable prototype of the function
7: //to create another method
8: MyClass.prototype.setName = function (value:String):void {
9:     //we have access to the property defined on MyClass object
10:    trace(this.name);
11:    this.name = value;
12:    trace(this.name);
13: }
14:
15: //create an instance
16: var myObject = new MyClass();
17: //accessing the method created earlier
18: myObject.setName("Joe");

```

Vou falar mais sobre as características dinâmicas do AS3 em uma seção posterior.

## Getters / Setters

Em qualquer linguagem OO geralmente usamos getters/setters para controlar as propriedades da classe que você deseja expor fora da classe. PHP é uma exceção. No entanto, em AS3 existe

um suporte especial para as propriedades usando o conjunto de palavras chave *get* e *set*. Aqui está um exemplo.

```
1: public class Employee {
2:
3:     private var _salary:int = 0;
4:     private var _income:int = 0;
5:
6:     function Employee() {
7:
8:     }
9:
10:    public function set salary(value:int):void {
11:        if (value > 0) {
12:            this._salary = value;
13:            this._income = this._salary * 12;
14:        }
15:    }
16:
17:    public function get salary():int {
18:        return this._salary;
19:    }
20:
21:    public function get income():int {
22:        return this.income;
23:    }
24: }
25:
26: //using this class
27: var emp:Employee = new Employee();
28: emp.salary = 1000;
29: trace(emp.income);
30: //this raise an error, because the income property is read-only
31: //for the outside code
32: emp.income = 120;
```

Basicamente, embora eu tenha um *set* e um *get* para o campo *\_salary*, posso chamar esses métodos, como se fossem propriedades, e não funções: *object.salary = 20* em vez de *object.salary(20)*. E se você optar por não definir um *set*, você torna a propriedade como somente leitura. Isto foi o que eu fiz com a propriedade *\_income*.

Esta característica, além de tornar o código um pouco mais limpo, torna mais fácil de escrever APIs ou classes que serão utilizadas em outras classes. Suponha que no meu exemplo eu escolhi criar o campo *\_salary* como um membro público. Se mais tarde eu decidir que eu preciso validar os valores que forem definidos, eu posso fazer isso no *set*. No PHP eu teria que fazer algo do tipo: *myObject.setSalary()*.

Em AS3 é possível iniciar a classe com a propriedade definida como: *public var salary:int* e quando você decidir que precisa de um *set*, você renomeia a variável e adiciona o seguinte método: *public function set salary()*. Qualquer código que usa a classe não será afetado por

esta mudança, porque ele ainda está acessando a variável com a mesma sintaxe:

```
objectInstance.salary = 10.
```

Esta uma convenção dizendo que se você usa *get* e *set* para definir propriedades, deve inserir um “\_” no nome da variável.

## Interfaces

Interfaces possuem a mesma implementação em PHP e AS4. A diferença mais notável é que no PHP você pode definir métodos e constantes, e em AS3, somente métodos. No entanto, você pode definir *gets/sets*:

```
1: public interface IEmployee {
2:
3:     public function set salary(value:int);
4:     public function get salary():int;
5:     public function get income():int;
6: }
```

## Exceções

Como no PHP, AS3 possui suporte para exceções:

```
1: try {
2:
3: } catch(e:Error) {
4:
5: } finally {
6:
7: }
8:
9: //throwing an exception
10: throw new Error("Some error");
```

*Error* é uma classe no mais alto nível das classes de erro existentes em AS3. Você pode criar suas próprias classes de erro, ou pode usar subclasses de erro existentes.

## Convertendo e testando o tipo de um objeto

Às vezes você precisa converter um objeto para um tipo diferente do existente, ou até verificar o seu tipo. Para fazer a verificação do tipo em PHP você usa *instanceof*, em AS3 você usa *is*. Para fazer uma conversão, em AS3 você pode usar duas diferentes sintaxes:

```
1: class A {};
2:
3: class B extends A {};
```

```
4:
5: var b:A = new B();
6: //casting
7: var c:B = b as B;
8: //or
9: var d:B = B(b);
10:
11: //checking the type of an variable
12: if (b is A)
13:     trace(true);
14: if (b is B)
15:     trace(true);
```

## Escopo de variáveis

Tendo visto como variáveis, funções e classes trabalham no Flex e no AS3, chegou a hora de falar de escopo de variáveis. No PHP, basicamente, você tem dois escopos: global (variáveis definidas no nível de arquivo) e local (variáveis definidas dentro de funções).

No Flex há cinco escopos possíveis: dentro da função, em um método de uma instancia, em um método estático, em uma classe e no escopo global. Somando-se estes modificadores de acesso (public/private/protected/interno) as coisas ficam um pouco mais complicadas em relação ao PHP.

Escopos podem ser aninhados, neste caso as variáveis / funções / membros do escopo podem ser aninhadas. Por exemplo, quando você declarar uma função anônima dentro de uma outra função, em AS3 todas as variáveis definidas na função mais externa estarão disponíveis na função mais interna. No PHP, você tem que passar as variáveis que você precisa usar, ou adicionar a declaração *use*:

```
1: //php code
2: function a() {
3:     $a = 1;
4:     $b = 2;
5:
6:     function b() use ($a, $b) {
7:
8:     }
9: }
10:
11: //AS3 code
12: function a():void {
13:     var a:int = 1;
14:     var b:int = 2;
15:
16:     function b():void {
17:         //variables a and b are available here
18:     }
19: }
```

Quando você declara uma função dentro de *Package* sem nome, ele é colocado no âmbito global e estará disponível em todo o código da aplicação. Entretanto, tudo que é declarado fora de um pacote ainda está no âmbito global, mas é visível apenas para o código do mesmo arquivo.

## Matrizes (arrays)

Matrizes no AS3 são muito semelhantes aos do PHP, com uma diferença: em uma matriz AS3 podemos ter apenas índices numéricos. Se você criar uma matriz associativa, você pode usar a classe *Object*. Se você deseja criar um “hash map” onde as chaves são objetos (e não strings) você pode usar a classe *Dictionary*. Você cria um array usando a classe *Array*, e você pode ter arrays multi dimensionais. Tanto para *Object* e *Array* você pode usar uma definição literal.

Vamos ver alguns exemplos:

```
1: var myArray1:Array = new Array(1, 2, "some string");
2: //creates an array with three elements: 0->1, 1->2, 3->some string
3:
4: //literal definition for an array
5: var myArray2:Array = [1, 2, 3];
6: //adding two more elements
7: myArray2.push(4,5);
8:
9: //a hash map, similar to associative arrays in PHP
10: var myMap:Object = new Object();
11: myMap.name = "Flex";
12: //literal definition of a map
13: var myMap2:Object = {name:"Flex"};
14:
15: //using Dictionary class
16: var dic:Dictionary = new Dictionary();
17: var anObject:Object = new Object(); //creating the key
18: dic[anObject] = "some value"; //adding a value to Dictionary
```

Você possui todos os métodos esperados para adicionar e remover elementos, incluindo push, shift, pop, unshift e splice. *Contact* pode ser usado para adicionar arrays em outras arrays. No exemplo anterior, você pode ver como eu usei o método *push* para adicionar mais dois elementos no array.

Arrays não tem comprimento fixo, pois eles podem crescer à medida que você adiciona mais elementos. No PHP, você pode usar “[]” para adicionar um novo elemento no final do array. Existe um método similar em AS3, que usa a propriedade *length* do array (você também pode usar a propriedade *length* para diminuir o tamanho do array):

```
1: var array:Array = new Array();
2: array[array.length] = 1; //array has the values: 1
```

```
3: array[array.length] = 23;//array has the values: 1, 23
```

Você pode usar *delete* para definir um elemento específico do array como *undefined*: *delete array [index]*. Isso não vai diminuir o comprimento da matriz. Você pode usar a instrução *for()* para percorrer um array usando sua propriedade *length*. Se você quiser percorrer um objeto (novamente isto poderia ser usado para criar algo semelhante a arrays PHP associativos) você pode usar o comando *for each* ou *for in* nas declarações (mais sobre isso na seção *Dynamic*).

## Namespaces

Se você está procurando um conceito equivalente em AS3 para os *namespaces* do PHP, você deve ler sobre a seção *Packages*, porque os *Packages* AS3 são semelhantes aos *namespaces* do PHP.

No ActionScript, *namespaces* tem um significado diferente. Vamos ver para que os namespaces são usados, e então dar alguns exemplos:

- Prevenir conflitos de nomes (você pode criar vários métodos com o mesmo nome na mesma classe, cada um em um *namespace* diferente)
- Marcar variáveis e métodos através de frameworks/programas para possuírem uma visibilidade customizada (por exemplo, Flex usa o namespace chamado *mx\_internal*; usando o namespace ao invés de *private* ou *protected*, torna possível a utilização destes métodos através de qualquer pacote ou classe a partir da estrutura do Flex. Ao mesmo tempo, desenvolvedores são alertados que estes membros não devem ser usados externamente, pois eles podem mudar)
- Implementar permissões de controle de acesso baseado em uma classe
- Criar uma classe que pode mudar seu comportamento baseado no namespace específico selecionado

Antes de entrar em detalhes, devo lembrar que namespaces são usados internamente pelo Flash Player para aplicar os modificadores de acesso: *public*, *protected*, *internal* e *private*.

Você define um *namespace* usando esta sintaxe: *identificador=URI*. Identificador é o que você vai usar para declarar variáveis/métodos, e quando você quiser qualificar um membro ou método, a fim de usá-lo. URI normalmente é uma URL que deve ser exclusiva para sua aplicação. Não tem de existir, e na maioria dos casos você pode usar o seu domínio. Por exemplo, eu poderia definir um namespace como este: *namespace online = "HTTP://corlan.org/apps/online"*.

Você pode definir um namespace onde as variáveis podem ser definidas: no nível mais alto de um *Package* (que está disponível em uma aplicação), ou no nível da classe (que só estaria

disponível na classe onde está definido). Ao nível de função você pode usar somente um namespace para o nome que foi definido em outro lugar (talvez seja para qualificar uma variável que foi definida em qualquer lugar usando o mesmo namespace; você precisa conhecer a URI para fazer isso).

Você pode declarar um método ou variável em um determinado namespace, colocando o identificador do namespace antes da declaração. Por exemplo: `var mynamespace a:int=1`. Quando você define uma variável ou método com um namespace, você não pode usar outros modificadores (como `private`, por exemplo).

Para chamar uma variável ou método que foi definido em um namespace, você usa o operador `::`. Suponha que você tenha definido um método chamado `myMethod()` em um namespace chamado `online`, você pode acessá-lo utilizando a sintaxe `objectInstance.online::myMethod()`. O mesmo vale para as variáveis. Às vezes você pode precisar usar muitas variáveis ou métodos que devem ser qualificados com o namespace. Você faz isso usando a diretiva `namespaceidentifier`. Por exemplo:

```
1: public function doSomething() {
2:     use namespace online;
3:     //call the method defined in that namespace:
4:     myMethod();
5: }
```

Você pode repassar namespaces, por exemplo, você pode retornar um namespace de um método, permitindo que o código chamado possa utilizá-lo para beneficiar um método ou membro.

Agora, vamos criar dois namespaces que podem ser usados para alterar o comportamento de uma classe em tempo de execução. Primeiro eu irei definir dois namespaces (vou usar um arquivo para cada namespace).

```
1: // ActionScript file online.as
2: package org.corlan {
3:     public namespace online = "http://corlan.org/apps/online";
4: }
```

```
1: // ActionScript file offline.as
2: package org.corlan {
3:     public namespace offline = "http://corlan.org/apps/offline";
4: }
```

Em seguida, vou utilizar estes dois namespaces para criar uma classe que persiste um objeto. Dependendo do estado de conexão podemos manter o objeto localmente (armazenando localmente, por exemplo), ou remotamente em um servidor (usando um serviço REST). A parte

interessante vem quando algum código precisa usar essa classe. O código de chamada não se preocupa com o método em tudo, ele só quer ter os objetos armazenados.

Usando estes dois namespaces, vou criar uma classe que tem dois métodos, chamados de *save()*, cada um definido pelo seu *namespace*. Depois, tenho uma variável privada que armazena o namespace atual a ser utilizado, dependendo do status de conexão com a Internet. O programa acessa o *namespace* atual usando um *get*, e usa isso para chamar o método *save()*. Novamente, a aplicação não sabe tudo sobre todos os métodos internos e não sabe sobre o namespace, e nem se importa com ele. Vamos ver o código *PersistObject*.

```
1: package org.corlan {
2:     import flash.events.Event;
3:
4:     public class PersistObject {
5:
6:         private var _mode:Namespace = offline;
7:
8:         public function PersistObject() {
9:
10:        }
11:
12:        online function save(object:Object):void {
13:            //save the object back to server
14:            trace("online");
15:        }
16:
17:        offline function save(object:Object):void {
18:            //save the object locally
19:            trace("offline");
20:        }
21:
22:        private function connectivityChanged(e:Event):void {
23:            //here the mode can be changed from offline to online
24:            //and vice-versa
25:        }
26:
27:        public function get mode():Namespace {
28:            return this._mode;
29:        }
30:    }
31: }
```

O próximo trecho de código utiliza esta classe. O código é simples e os comentários são auto explicativos.

```
1: //creating an object that we want to be stored
2: var object:Object = {book:"Ulysses", author:"James Joyce"};
3: //create an instance of PersistObject
4: var persistenceObject:PersistObject = new PersistObject();
5: //get the current namespace
6: var currentMode:Namespace = persistenceObject.mode;
```

```
7: //use the namespace we retrieved to qualify the save method()
8: persistenceObject.currentMode::save(object);
```

## Acessibilidade no *Namespace*

Você pode usar o mesmo acesso aos modificadores que usa para variáveis ou métodos: *public*, *internal*, *protected*, *private* (por namespaces definidos no nível do pacote você pode usar somente *publics* ou *internal*). Combinando isso com a localização de um *namespace* você tem maior controle sobre a visibilidade de um *namespace* de uma aplicação.

## Trabalhando com XML

No PHP existe amplo suporte ao XML, através de funções nativas ou extensões adicionais. Em AS3 existem duas classes que representam XML nativamente: *XML* e *XMLList*. AS3 implementa a classe XML baseado no DOM W3C (você tem método como *children()*, *appendChild()*, *parent()*, *insertChildBefore()* e assim por diante). Quando você trabalhar com XML é uma boa idéia saber usar E4X. E4X (ECMAScript for XML) é uma extensão de linguagem ECMA-262 implementada pelo ActionScript. Você usa XML para representar um documento XML. Qualquer nó do documento é convertido para um *XMLList* mesmo quando existe apenas um único nó filho.

Você pode criar um objeto XML usando qualquer um dos seguintes métodos:

- Escrever XML utilizando o formulário literal
- Criar uma instância de XML e depois importar o XML de um arquivo externo
- Criar uma instância de XML e usar a notação para adicionar/remover uma estrutura:

```
1: var author:XML = <author/>;
2: author.@id = 1; //setting an attribute called id and its value
3: //adding two child nodes to author:
4: author.name = "Mihai Corlan";
5: author.article = "Flex for PHP developers";
6:
7: //this code is equivalent with:
8: var author:XML = <author id="1">
9: <name>Mihai Corlan</name>
10: <article>Flex for PHP developers</article>
11: </author>;
```

Usando E4X você pode facilmente encontrar nós através da criação de condições baseada no nome dos nós ou do valor dos atributos. Você pode usar o operador descendente “..” para

todos os nós dado um nome (por exemplo, para recuperar todos os nós de um programa você pode escrever: `programs..program`). Você pode criar condições baseadas em atributos usando o operador "@" (por exemplo, `programs..program.(@id==2)`). E finalmente, usando a notação de ponto você pode navegar através dos nós (tenha em mente que qualquer nó filho é tratado como uma XMLList, quando ele é um único nó).

Abaixo você pode ver exemplos de como usar E4X para trabalhar com XML.

```
1: var programs:XML = <root>
2:   <program id="1">
3:     <name>Flex</name>
4:   </program>
5:   <program id="2">
6:     <name>ActionScript 3</name>
7:   </program>
8:   <program id="3">
9:     <name>AJAX</name>
10:  </program>
11: </root>;
12:
13: //retrieving the second program node and printing its name
14: trace(programs.program[2].name[0]);
15: //retrieving all the program nodes:
16: var list:XMLList = programs..program;
17: //retrieving all the program nodes that have an id attribute equal to 2
18: var list:XMLList = programs..program.(@id==2);
```

## Dynamic Actionscript

Lembra-se da definição de AS3? Na definição afirmei que AS3 é uma linguagem de scripts dinâmica. Vamos nos aprofundar um pouco mais neste recurso. Dinâmica significa que um objeto pode ser modificado em tempo de execução, adicionando ou removendo os métodos e membros. É possível adicionar novos métodos para a própria classe (e qualquer objeto criado a partir desta classe terá os métodos). Você pode até mesmo criar novas classes a partir do zero (usando a propriedade *prototype*). Em AS3 existem objetos internos dinâmicos, como *Object*, e em Flex há um outros exemplo, *ObjectProxy*.

Se você está se perguntando por que este recurso existe, em primeiro lugar, a resposta é simples: as versões anteriores do ActionScript não tem todas as características que a AS3 OO oferece hoje. Eu tenho que dizer, pela minha experiência, muitos desenvolvedores não usam os recursos dinâmicos do AS3. Existem algumas razões para isso. Primeiro, o tempo de acesso para os membros dinâmicos são mais lentos do que os membros fixos. Segundo, você deixa um código mais propenso a erros (não há nenhuma verificação de erro em tempo de compilação, por exemplo).

Você não está limitado a classes dinâmicas pré fabricadas, pode criar objetos dinâmicos através do modificador *dynamic* na definição da classe:

```
1: dynamic public MyDynamicObject {  
2:  
3: }
```

Agora, usando a classe definida, você pode adicionar membros em tempo de execução (lembre-se que todas as variáveis dinâmicas não possuem tipo e são públicas):

```
1: var a:MyDynamicObject = new MyDynamicObject();  
2: a.author = "Mihai Corlan";
```

Você pode percorrer todos os membros de uma classe dinâmica usando o loop *for-in*:

```
1: for (var memberName:* in a) {  
2:     trace(memberName); //outputs author  
3:     trace(a[memberName]); //outputs Mihai Corlan  
4: }
```

## Flex é assíncrono

Até agora, eu falei de muitas características do Flex, e muitas deles foram bastante semelhantes com o que falei sobre PHP. Entretanto, a natureza assíncrona do Flex é algo bastante diferente de qualquer coisa em PHP. É importante entender isso, parar de lutar contra isso, e seguir o fluxo correto.

O que significa que o *Flex* é assíncrono? Suponha que você criou uma aplicação em Flex, e depois da aplicação estar carregada no navegador, o usuário pode escolher carregar figuras de outro site. Você pode usar a classe *URLLoader* para esta tarefa. Quando você executa o método *load()*, na próxima linha de código você ainda não vai ter dados. O fluxo de script não pausa após o *load()* e aguarda os dados da imagem serem carregados. Em vez disso a execução do script é retomada. Como programador você lida com esta natureza assíncrona usando os recursos internos do AS3 de eventos. Se você estiver familiarizado com programação em AJAX, isto é muito semelhante a uma chamada em AJAX: você fornece uma função chamada “callback” que será chamada somente quando os dados forem carregados.

Voltando ao exemplo do *URLLoader*, vou deve adicionar um evento que irá “escutar” o evento *result*. Esta é a função que será chamada uma vez que os dados estão carregados. Aqui esta um exemplo de como o código deve ser:

```
1: function loadPic():void {  
2:     var loader:URLLoader = new URLLoader();  
3:     loader.dataFormat = URLLoaderDataFormat.BINARY;
```

```
4: //adding the event handlers or listeners
5: loader.addEventListener(EventComplete, picLoaded);
6: loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR, picError);
7: //starting the loading
8: loader.load(new URLRequest("http://some_url"));
9: }
10:
11: //event handler for
12: function picLoaded(event:Event):void {
13: //get the data from the loader object
14: //use the target property to get the loader object
15: (event.target as URLLoader).data;
16: }
17:
18: //event handler for the error event
19: function picError(event:IOErrorEvent):void {
20: //displays the error id in a pop-up window
21: Alert.show(event.errorID);
22: }
```

Eu posso resumir isso como: não o chame, deixe-o te chamar.

Como eu disse, AS3 já vem preparado para manipular eventos. A classe principal de todos os eventos é *Event*. Todos os objetos que trabalham assincronamente tem o método *addEventListener()*, e os dois primeiros argumentos são o tipo de evento e o nome da função que será chamada quando o evento ocorrer. Você pode pensar que somente objetos que trabalham com recuperação de dados estão sujeitos a utilização de eventos. Atualmente este não é o caso; todos os componentes ou objetos visuais também possuem eventos. Por exemplo, toda aplicação Flex possui o evento *creationComplete*. Este evento é acionado uma vez que todos os componentes da aplicação estão prontos e desenhados na tela.

Embora você possa sentir que esse código não é tão simples como PHP, existe uma boa razão para ter chamadas assíncronas espalhadas pelo Flex (e o Flash Player). Flex é uma tecnologia *client-side*. Se todas as chamadas fossem sincronizadas, a interface do usuário da aplicação deixaria de se tornar sensível para qualquer chamada que envolvesse carregamento de dados. E os usuários odeiam interfaces em que o sistema para de responder.

Você pode cancelar alguns eventos e até mesmo alterar o comportamento padrão. Vou deixar para você explorar esses detalhes quando for preciso; por enquanto você deve ter uma boa idéia sobre *events* e sobre *eventListeners*.

## DataBind, metada tags e reflection

DataBind é outra funcionalidade do Flex que torna a vida do desenvolvedor bem mais fácil, e ao mesmo tempo reduz a quantidade de linhas de código. DataBind é uma forma elegante de ligar os dados de um modelo a uma camada de layout e automaticamente alterar os dados tanto no objeto quanto no layout.

Já que o Flex é usado para criar interfaces e aplicações, componentes Flex normalmente devem exibir uma grande quantidade de dados. Quando os dados são modificados, mesmo em tempo real, normalmente você deseja exibir os dados alterados mais recentes e são os dados antigos. Utilizando *dataBind* você consegue isso com muita facilidade. *DataBind* ligam a propriedade de um objeto (chamado de *source*) para a propriedade de outro objeto (chamado de *destination*), então sempre que a origem muda, o destino é automaticamente atualizado.

Em Flex 4 existe um suporte para uma ligação bidirecional (atualmente pode ser feito também em Flex 3, mas necessita de uma declaração como um segundo passo), no qual significa que opera no sentido inverso também: quando o destino é alterado, os novos valores são copiados para a origem. Isto é útil quando temos dados provenientes de um modelo e de um formulário. Você liga os dados no formulário, e quando o usuário alterar valores no formulário, os dados irão atualizar o modelo de acordo com os valores do formulário. É hora de ver algum código:

```
<?xml version="1.0" encoding="utf-8"?>
  2: <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="horizontal">
  3:   <mx:Script>
  4:     <![CDATA[
  5:
  6:       [Bindable]
  7:       private var labelValue:String = "Hello World!";
  8:
  9:       private function add():void {
 10:         labelValue += "!";
 11:       }
 12:     ]]>
 13:   </mx:Script>
 14:   <mx:Label id="myLabel" text="{labelValue}"/>
 15:   <mx:Button label="Add a !" click="add()"/>
 16: </mx:Application>
```

O metadados *Bindable* na variável *labelValue* marca ele como uma fonte de dados. Em seguida, eu usei “{ }” para o atributo *text* de um *Label* para marcar a propriedade como destino de uma ligação (*binding*). Feito o *binding*, a qualquer hora que a variável *labelValue* for alterada, o *Label* atualiza a propriedade *text* para refletir a mudança. Eu poderia usar a mesma variável para muitos *Labels* ou *TextInputs*, e todos eles poderiam ser atualizados para refletir o novo valor.

Há também uma sintaxe MXML: `<mx:Binding source="labelValue" destination="myLabel.text"/>`

Se você precisar ligar o dado a um controle de edição (por exemplo, o *TextInput*) e copiar o valor de volta para a fonte de dados, no Flex 4 você pode usar o operador bidirecional “@”:

```
1: <s:TextInput id="txt" text="@{labelValue}"/>
```

E se você quiser usar a tag *Binding*, você seta o atributo *twoWay* como *true* (novamente isto somente é possível no Flex 4).

```
1: <mx:Binding source="labelValue" destination="myTextBox.text" twoWay="true"/>
```

A fim de implementar o *dataBinding*, o Flex adiciona código em tempo de execução (lembre que *dataBinding* é uma funcionalidade do Flex e não do Flash Player ou do AS3), e a beleza disso é que você não precisa ficar escrevendo código.

Embora o *dataBinding* ofereça uma maneira simples de ligar um modelo de dados a uma camada de exibição, haverá um impacto no desempenho se você tem várias ligações para as variáveis onde são atualizadas dezenas ou centenas de vezes por segundo. Para essas variáveis, não há necessidade de atualizar a interface do usuário muitas vezes, porque não há um limite de frames por segundos impostas pelo próprio navegador (algo em torno de 50 frames por segundo). Como resultado, é bastante inútil mostrar centenas de mudanças por segundo em tempo real.

Outro aspecto para ter em mente é que nem todos os objetos são *bindable*. Por exemplo, *Object* e *Array* não são *bindable*, você deve usar *ObjectProxy* e *ArrayCollection*. Quando você cria classes para o seu modelo de dados, se você quiser que todos os seus membros sejam *bindable*, você pode inserir a *metadata [Bindable]* acima da definição da classe, ao invés de adicionar para cada propriedade:

```
1: package org.corlan {
2:
3:     [Bindable]
4:     public class VOAuthor {
5:
6:         public var id_aut:int;
7:         public var fname_aut:String;
8:         public var lname_aut:String;
9:     }
10: }
```

Agora vamos ver mais as *tags* de *metadata* (as vezes chamadas de *annotations*). Você já viu essas tags quando viu o *[Bindable]*. Para obter uma lista completa de tags, clique aqui. Em alguns casos, as tags de metadata são utilizados pelo compilador MXML, a fim de gerar código (como no *Bindable*), em outros casos, você pode usar tags para sinalizar a IDE Flash Builder ou para criar propriedades no código MXML. Este é o caso do metadata Event. Por exemplo, suponha que eu escrevi uma classe que dispara um evento quando o filme está carregado. Eu posso usar o metadata Event para declarar o tipo de evento e o seu nome. Fazendo isso, eu posso usar a propriedade *movieLoadedEvent* na tag MXML *MovieLoader* para registrar um event listener para este evento. Vamos ver o código da classe e como você pode ver a sua utilização no MXML:

```

1: //class definition
2: package org.corlan {
3:     import flash.events.EventDispatcher;
4:     import flash.events.IEventDispatcher;
5:
6:     [Event(name="movieLoadedEvent", type="flash.events.Event")]
7:
8:     public class MovieLoader extends EventDispatcher {
9:
10:         public function MovieLoader(target:IEventDispatcher=null) {
11:             super(target);
12:         }
13:
14:     }
15: }

```

```

1: <?xml version="1.0" encoding="utf-8"?>
2: <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
3:     xmlns:local="org.corlan.*" layout="horizontal">
4:     <mx:Script>
5:         <![CDATA[
6:             private function movieLoadedListener(event:Event):void {
7:                 //do something with it
8:             }
9:         ]]>
10:    </mx:Script>
11:
12:    <local:MovieLoader id="loader" movieLoadedEvent="movieLoadedListener(event)"/>
13: </mx:Application>

```

Existem outras coisas interessantes que você pode fazer com metadados. Se você definir uma *flag* no compilador (mantenha metadata AS3 seguida do nome da tag), você pode adicionar seu metadata personalizado e o compilador irá colocar a tag no bytecode da aplicação. Você pode usar estas tags em tempo de execução. Por exemplo, neste [post](#), você pode ler como você pode usar um metadata personalizado para fornecer uma forma de persistir um modelo de dados constantes do pedido de um Adobe AIR (leia mais sobre Adobe AIR um pouco mais tarde).

E isto me leva ao último tópico desta seção: *Reflection*. A fim de utilizar os metadados personalizados que você tem que utilizar a API do AS3 sobre *reflection*.

No PHP há uma completa API para utilização de *reflection*. *Reflection*, *ReflectionFunction*, *ReflectionParameter*, *ReflectionMethod* e assim por diante. Aqui está um exemplo usando a classe de *reflection* em uma classe PHP muito simples.

```

class SimpleClass {
2:
3:     public $public = 'Public';
4:     protected $protected = 'Protected';
5:     private $private = 'Private';
6:

```

```
7:     private function SimpleClass() {
8:         echo('SimpleClass() called');
9:     }
10:
11:     private function __construct() {
12:
13:     }
14:
15:     function displayVar() {
16:         echo "SimpleClass class\n";
17:     }
18: }
19:
20:
21: Reflection::export(new ReflectionClass('SimpleClass')); //this outputs:
22:
23: Class [ <user> class SimpleClass ] {
24:     @@ /Applications/MAMP/htdocs/_learning/classes.php 7-26
25:
26:     - Constants [0] {
27:     }
28:
29:     - Static properties [0] {
30:     }
31:
32:     - Static methods [0] {
33:     }
34:
35:     - Properties [3] {
36:     Property [ <default> public $public ]
37:     Property [ <default> protected $protected ]
38:     Property [ <default> private $private ]
39:     }
40:
41:     - Methods [3] {
42:     Method [ <user> private method SimpleClass ] {
43:         @@ /Applications/MAMP/htdocs/_learning/classes.php 13 - 15
44:     }
45:
46:     Method [ <user, ctor> private method __construct ] {
47:         @@ /Applications/MAMP/htdocs/_learning/classes.php 17 - 19
48:     }
49:
50:     Method [ <user> public method displayVar ] {
51:         @@ /Applications/MAMP/htdocs/_learning/classes.php 22 - 25
52:     }
53:     }
54: }
55:
```

Em AS3 existem três funções no pacote *flash.utils*, que pode ser usado para *Reflection*: *describeType()*, *getDefinitionByName()* e *getQualifiedSuperClassName()*. Aqui está um exemplo da saída de *describeType()* (a saída é um XML):

```
1: public SimpleClass classe (
2:
3: public var _public: String = "Público";
4: protected var _protected: String = "Protegido";
5: private var _private: String = "Private";
6:
7: SimpleClass function () (
8: trace ( "SimpleClass () chamado");
9:)
10:
11: displayVar função pública (): void
12: (
13: trace ( "class SimpleClass");
14:)
15:)
16:
17: função refletir (): void (
18: var s: SimpleClass SimpleClass = new ();
19: var description: XML = describeType (s);
20: trace (descrição);
21:)
22:
23: // saída:
24: <type name= base "org.corlan::SimpleClass" = "Object" isDynamic= "false" isFinal= "false"
isStatic= "false">
25: <extendsClass type= "Object" />
26: <method name= "displayVar" declaredBy= "org.corlan::SimpleClass" returnType= "void" />
27: <variable name= "_public" type= "String" />
28: </ type>
```

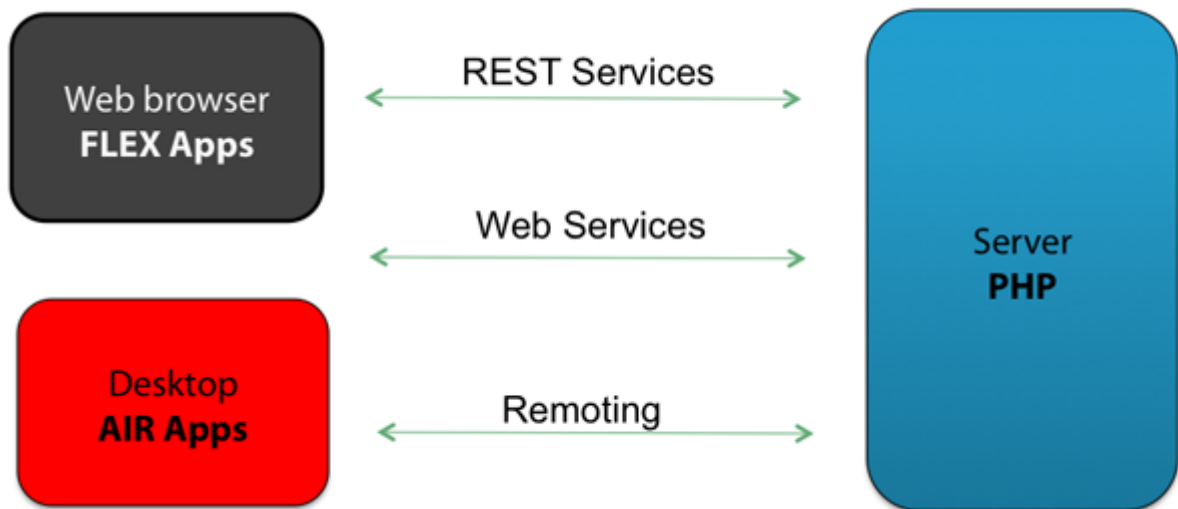
## Onde estão meus dados? Traga-os pra mim!

Como um desenvolvedor PHP, você tem uma maneira muito direta para ler dados, analisá-los e exibi-los na tela. Conectar-se a um banco de dados MySQL é a primeira coisa que qualquer programador PHP aprende. Na verdade, eu duvido que você conseguiu ler este artigo inteiro até aqui sem dar uma olhada nessa seção.

Mas e o Flex? Eu tenho que desapontá-lo, porque você não tem acesso direto aos dados armazenados em um banco de dados. Mas há algo de bom nisso, eu acho, porque você pode continuar a escrever arquivos PHP para ler/gravar dados em seu banco de dados, mesmo quando estiver criando uma aplicação em Flex. Porque não há uma maneira direta de ler dados de um banco de dados? Por causa do velho ditado: "Você nunca deve confiar no cliente!". Suponha que o cliente (o browser) é um componente Flex que sabe como se conectar ao servidor MySQL. Como você armazena as credencias de modo que não seria fácil

roubá-los para ter o banco de dados comprometido? Esta é apenas uma das razões porque não é uma boa idéia ter uma tecnologia que roda no lado do cliente poder-se conectar no banco de dados, sem usar um servidor de aplicações entre eles.

Basicamente, em aplicações Flex você confia aos scripts *server-side* (lado do servidor) para que estes acessam os dados. Flex oferece uma forma de chamar páginas do servidor e aguardar pela sua resposta. Existem três maneiras diferentes de se conectar a uma fonte de dados no lado do servidor: REST, WebServices e *Remoting* (RPC).



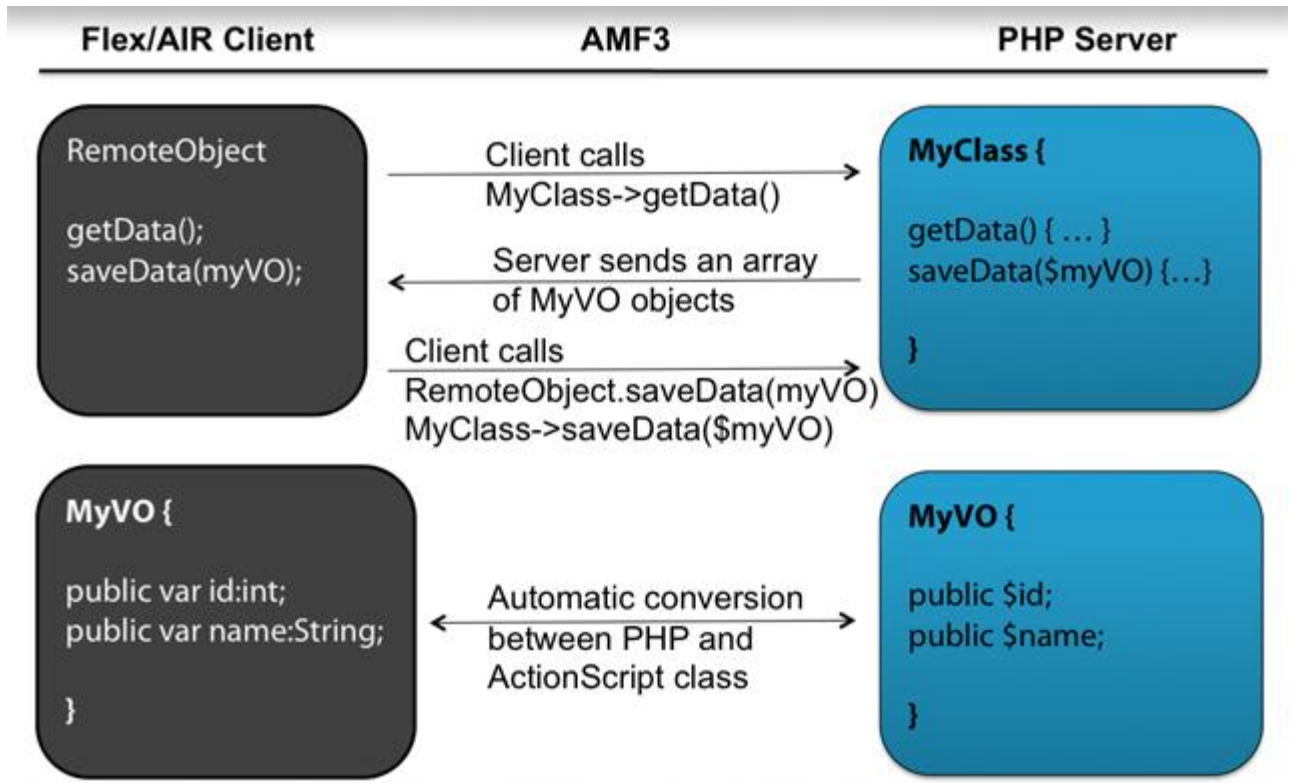
Você pode usar a classe `HTTPService` para utilização dos serviços REST. Você pode enviar variáveis POST quando faz uma requisição ao servidor, e a resposta pode ser XML, Json (existe uma biblioteca de terceiros para manipular JSON), ou uma formatação personalizada.

Se você tem WebServices no servidor, você pode usar a classe `WebService`.

Mas o método mais interessante é o *Remoting* (usando a classe `RemoteObject`). Há três razões porque eu acho que é o melhor método. Primeiro, usando *Remoting*, você pode utilizar qualquer classe PHP que você tem no seu servidor, chamado através de um método público. Basicamente, a partir do Flex, você usa uma instância do `RemoteObject`, como se fosse a classe PHP remota. Segundo, você pode mapear um modelo de dados no lado do PHP para um modelo de dados `ActionScript`, e pode fazer a conversão automaticamente. Isto é extremamente importante, porque quando você usa objetos tipados, você obtém os benefícios da compilação para encontrar erros, ao invés de encontrá-los em tempo de execução. Em terceiro lugar, o formato de mensagens para esta forma de comunicação, AMF3 (`Action Message Format`) é um formato binário, que pode ser muito mais rápido e menos, em comparação com SOAP/XML/JSON, especialmente para grandes conjunto de dados. O formato em si é aberto, e qualquer um pode ler a documentação e implementar como utilizá-lo.

AMF3 é mais rápido porque ele codifica dados. Por exemplo, se a mesma seqüência é repetida em um conjunto de dados, então é codificado uma vez, e todas as outras ocorrência desta

seqüência se tornam referencias. Se um número é menor do que quatro bits, então, apenas o número mínimo de bytes exigido são utilizados.



James Ward da Adobe criou um ótimo [teste](#) que mostra as diferenças entre os métodos de comunicação.

Remoting é suportado no Fle, porém no lado do servidor a história não é a mesma. O PHP não suporta remoting e AMF3 nativamente. É por isso que você precisa de uma biblioteca do lado do servidor para habilitar o remoting para um servidor PHP. Há quatro bibliotecas disponíveis, todas são livres, e eu escrevi tutoriais para usar cada um deles: [Zend AMF](#), [PHPAMF](#), [WebORB for PHP](#), [SabreAMF](#). E você pode ler aqui um [post](#) que compara cada um deles.

Devido ao tipo de dados nativos serem convertidos automaticamente (tipo PHP para o tipo AS3 e vice versa), você tem que prestar atenção com os tipos nativos de cada linguagem. [Aqui](#) há um exemplo de conversão de tipos na biblioteca AMFPHP.

### Autenticação de usuários em projetos Flex e PHP

Então, como a autenticação de usuários pode ser feito em projetos Flex e PHP? A resposta é muito simples, valide apenas em sites PHP, você usa uma sessão e uma forma de validar o usuário e a senha.

Basicamente, sempre que uma chamada é feita a partir do Flex, o id da sessão é automaticamente anexado. Assim, se o usuário foi autenticado anteriormente, a mesma sessão será utilizada.

Saiba mais sobre este assunto [aqui](#).

## Trabalhando em projetos Flex e PHP

Felizmente, tecnologias como PHP e Flex são maduros e , assim, possuem uma abundância de opções quando se trata de ferramentas. Vou apresentar alguns deles nesta seção

## Editores de texto para Flex e PHP

A primeira opção, pode-se considerar a utilização livre do Flex SDK. Especialmente se você ama linha de comando e editores de texto como o vi. Você pode escrever o código em seu editor favorito e usar a linha de comando ou alguma ferramenta para compilar/depurar a sua aplicação.

## Flex Builder / Flash Builder e Eclipse PDT / Zend Studio

Eu prefiro usar uma IDE mais moderna. Para Flex e projetos PHP, provavelmente a melhor combinação é o Flex Builder ou Flash Builder 4 e o Zend Studio. O Flex Builder é o nome da IDE da Adobe até a terceira versão, a quarta versão foi renomeado para Adobe Flash Builder. Esta IDE é baseada no Eclipse, está disponível para Windows e MAC OS, e vem como um plug-in para versões independentes. Por exemplo, se você tem Zend Studio, você pode considerar o Flash Builder plugin e instalar-lo por cima do Zend Studio. Você pode usar o Flex Builder 3 por 60 dias (versão Trial) e o Flash Builder 4 está em [beta](#) (2009). Se você é um professor ou estudante pode obter uma [licença livre](#).

Se você preferir trabalhar com o Eclipse PDT, você pode usar a mesma abordagem: instalar a versão plug-in do Flash Builder 4 ou o contrário, instalar a versão plugin do eclipse PDT sobre o Flash Builder.

Flash Builder 4 oferece assistentes para trabalhar com PHP e Flex: pode-se examinar o código PHP e gerar código AS3 e gerar códigos em Flex ([aqui](#) esta um tutorial sobre este assunto). Você pode usá-lo para debugar. Edir performance (profile), compilar e executar aplicações (web ou AIR). Você pode exportar uma aplicação como *release*, e existe suporte para refatoração de código bem como um modo de design e editor de *states*.

Também se integra ao Flash Catalyst, para que possa criar a interface de uma aplicação em Flash Catalyst e então abrir o projeto gerado no Flash 4 e continuar a criar a aplicação definido

lógicas de negócio (você pode assistir a este [screen cast](#) para ver como criar um vertical scrollbar usando Adobe Illustrator, Flash Catalyst e Flash Builder 4).

Aqui estão algumas outras IDEs (comerciais) para flex: IntelliJ IDEA e FDT (baseado no Eclipse).

## Depuração de aplicações FLEX

Você pode depurar o código do Flex usando o depurador do Flex SDK, ou o depurador do Flash Builder 4 (ou o Flex Builder 3). Se você escolheu um conjunto de IDEs Flash Builder 4 e Zend Studio, então você pode depurar o código PHP e o Flex no mesmo projeto com muita facilidade. Você faz uma chamada do Flex para o PHP e então entra no depurador PHP, então, quando a resposta está de volta, você entra no depurador Flex. [Aqui](#) e [aqui](#) estão alguns vídeos sobre o tema, e [aqui](#) temos um tutorial sobre depuração com Eclipse PDT e Flex Builder.

No PHP, uma das primeiras abordagens que eu uso se tiver erros é usar a combinação de *die()* e *var\_dump()* para ver o que está acontecendo. Em AS3 você pode usar o *trace()* para exibir o valor de variáveis na janela console. Este é um caminho não obstrutivo para exibir informações; você pode também usar a classe *Alert* para exibir mensagens em uma janela popup (como no debug Javascript antes do firebug).

O elemento chave para manter a mente clara é: agora, você tem um cliente separado do servidor, e os problemas podem estar no cliente, no servidor, ou na rede.

## O que é Adobe Air

O Adobe AIR é um *runtime* desktop para aplicações RIA, que funciona como aplicações desktop no Windows, Mac e Linux. Com o Air, você pode criar um único aplicativo que pode ser executado em qualquer um desses sistemas operacionais. Exemplos de aplicações AIR: [Tour de Flex](#), [web](#), [Times Reader](#), [ferramentas Dojo](#), e [Sideline do Yahoo!](#).



Você poderia pensar em Adobe AIR como “Flash Player” para desktop. No entanto, o Adobe AI é mais que apenas um “Flash Player”.

Dentro do runtime, existe um mecanismo de HTML (WebKit, o mesmo motor usado pelo Safari e Google Chrome) e uma *engine* modificada do Flash Player. Estas duas *engines* oferecem um conjunto de APIs que dão acesso à runtime de um aplicativo AIR. Há APIs para escrita/leitura

de arquivos no disco, detecção de conectividade de rede, detectando o número de monitores conectados e resolução, atualizações de aplicações, notificações no desktop, base de dados locais, arrastar e soltar e muito mais.



Como um desenvolvedor web você pode escolher qualquer combinação das seguintes tecnologias: Flex, ActionScript 3, ou HTML / CSS / Javascript. Está certo, você pode criar aplicações em AIR usando somente HTML, Javascript e CSS. Na verdade, o toolbox DOJO e o SideLinedo Yahoo são criados usando HTML/CSS/JS.

Então, com AIR você pode aproveitar seus diversos recursos para construir uma aplicação desktop. Mas porque você quer criar uma aplicação web que funciona como um aplicativo desktop? Na verdade, existem muitas razões para isso:

- Você quer usar a aplicação, ou partes dela, quando não estiver conectado na Internet
- Você não quer usar um navegador como o chrome, e criar o seu próprio
- Você quer persistir arquivos na máquina do usuário
- Você quer criar um sistema de notificação, e deseja executar a aplicação minimizada na bandeja do sistema (por exemplo, mensageiros instantâneos podem notificá-lo quando uma nova mensagem é recebida, embora estejam minimizados e sem foco)

Para desenvolver aplicações em AIR, você pode usar gratuitamente o AIR SDK (você tem ferramentas de linha de comando para a construção, teste e depuração), você pode usar o

Aptana Studio (se quiser criar aplicações AIR com HTML / JS / CSS) ou você pode usar o Flash Builder 4 (ou Flex Builder 3).

Finalmente, qualquer aplicação Flex que foi criada para o navegador pode ser transformado em uma aplicação AIR a qualquer momento.

## O que vem a seguir?

No início do próximo ano, Flex 4 será lançado. A Adobe desenvolveu uma nova ferramenta (que ainda está em beta) chamado Flash Catalyst que pode ser usado para transformar desenhos estáticos criados no Photoshop ou Illustrator em interfaces para o Flex. Imagine uma ferramenta que poderia ter como entrada um arquivo no *Photoshop* ou *Illustrator*, e a saída de HTML / CSS / Javascript e manter o mesmo visual dos arquivos originais. Isto é o que o Flash Catalyst está fazendo, só que as saídas são em Flex 4 e não HTML.

Ao mesmo tempo, estamos nos concentrando muito em fazer a plataforma Flash estar disponível em todas as telas: a partir de computadores e dispositivos portáteis, a partir de telas para TVs. Agora temos o Flash Lite 3, disponível em celulares (Nokia, Sony Ericsson, HTC, Android, Palm). No próximo ano, lançaremos a versão móvel do Flash Player 10. Além disso, é muito possível que no próximo ano vamos ter a televisão pela primeira vez com suporte para Flash. Como seria legal ter a interface com o usuário que aproveita todos os recursos do Flash Player, ou para poder assistir vídeos de alta definição a partir da Web (Flash tem o suporte ao padrão H-264). Alguns analistas acreditam que no futuro próximo haverá mais telefones móveis conectados à Internet do que computadores, e muitas pessoas vão usar seus dispositivos móveis como o principal meio de acesso a conteúdos de Internet.

O que isso significa para um desenvolvedor web? Isso significa que você pode expandir sua área de especialização e que você tem a oferecer a partir de sites web para desktops RIAs (usando Adobe AIR), e de computadores para celulares e outros dispositivos com telas. Claro, a fim de fazer isso, você precisa retocar algumas habilidades, mas o nível de complexidade não é nem perto do que é preciso para se tornar bom em uma plataforma como c ou c++.

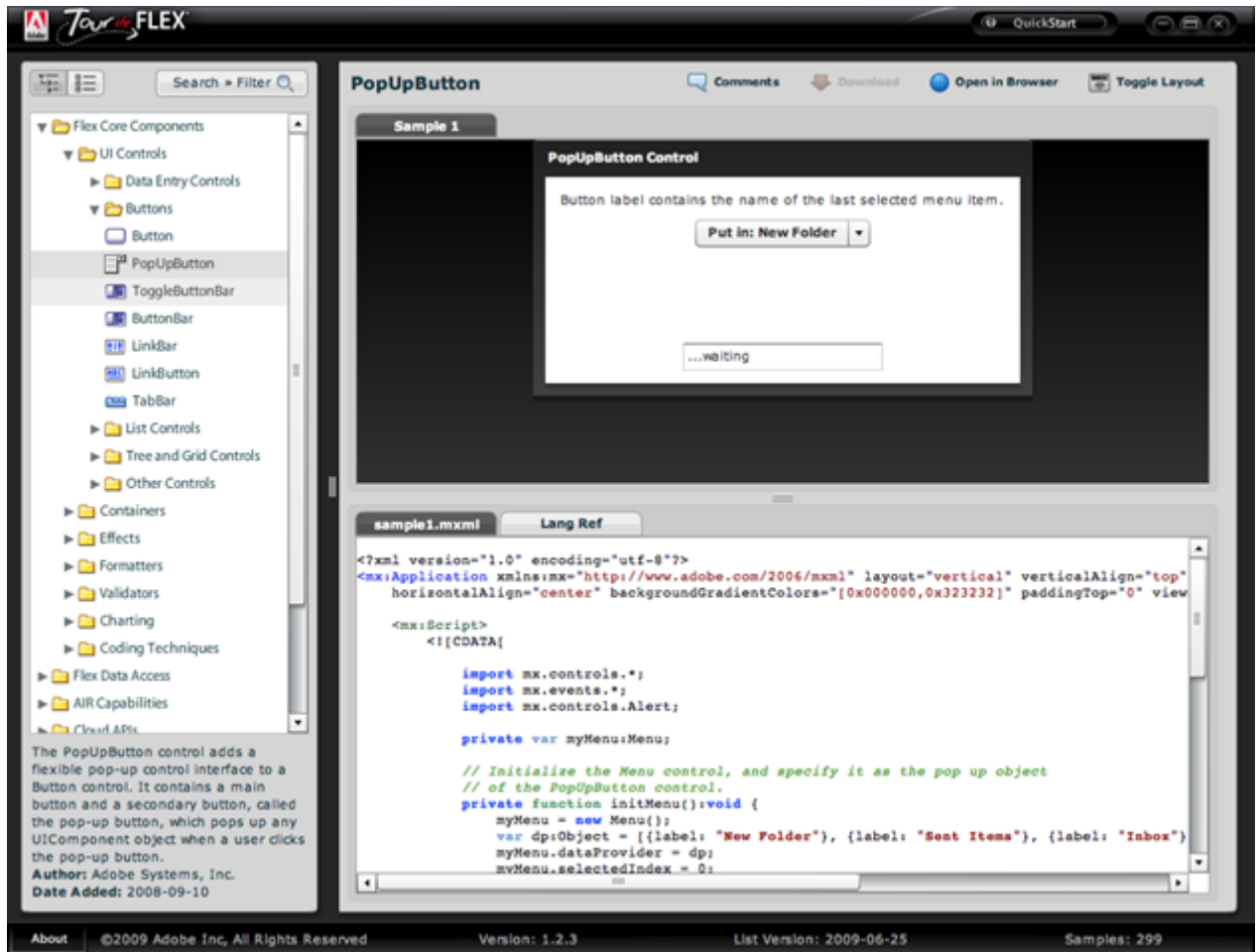
## Para onde ir agora?

Espero que você tenha encontrado respostas para algumas de suas perguntas. Se você pensa sério sobre como entrar no mundo Flex, aqui estão alguns recursos:

### Tour de Flex

Eu vejo o Tour de Flex a versão RIA do meu amado [php\\_manual.chm](#). Você pode instalá-lo [aqui](#). Ele dá exemplos de como usar os componentes Flex (você pode ver como funciona

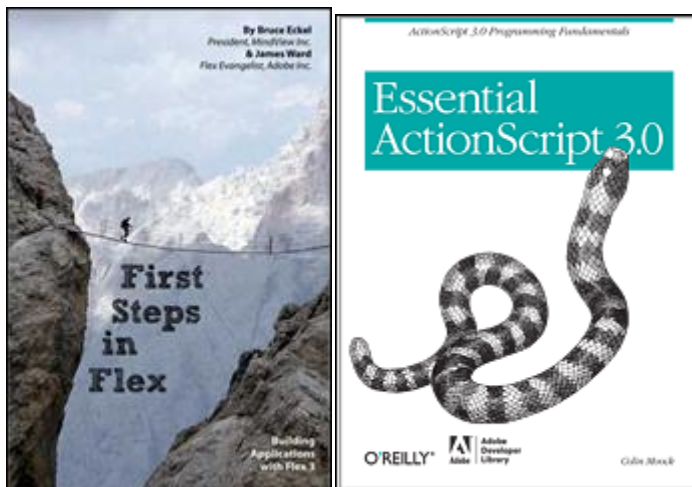
juntamente com o código). Há também um fácil acesso à documentação Flex. O próprio aplicativo é criado usando Flex e Adobe Air.



## Livros

Existem muitos livros sobre Flex e ActionScript 3. No entanto meus favoritos pessoais, e aqueles que eu recomendo são:

- [First steps in Flex](#): por Bruce Eckel e James Ward (Evangelista técnico da Adobe); este livro traz uma introdução de como usar o Flex para desenvolver aplicações web e desktop, e você pode read em um final de semana.
- [Essential ActionScript 3](#): por Colin Moock; este é um livro bem grande e você tem que ser um excelente leitor para ler em um final de semana. Apesar de não cobrir Flex, você pode aprender quase tudo sobre ActionScript 3.



## Websites

Existem centenas de sites e blogs que cobrem Flex e ActionScript 3. É impossível lhe dar uma lista completa, mas eu irei exibir os meus favoritos:

- [Adobe Developer Connection](#). Um ótimo lugar para ler artigos técnicos sobre Flash,, Flex e AIR. É atualizado semanalmente. Há uma [seção](#) dedicada ao Flex e PHP.
- [Adobe TV](#): Este site dá acesso a toneladas de vídeos com apresentações de conferências e tutoriais em vídeo.
- [Gotoandlearn.com](#): Uma excelente coleção de tutoriais em vídeo pelo colega evangelista Lee Brimelow.
- [Documentação do Flex](#): Tenha em mente que você pode baixar um ZIP com esta documentação. Você pode encontrar [aqui](#) recursos em vários formatos.
- [Grupos da Adobe](#). Uma plataforma livre onde você pode encontrar grupos de usuários Aodbe (incluindo Flex ou Flash Groups). Dê uma olhada, pode haver um grupo de usuários Flex em sua região, e geralmente, eles organizam encontros mensais. Existem fóruns sob esta plataforma, e suporte à localização, então não há conteúdo somente em inglês.
- Meu blogroll: Acesse-o, a maioria deles são de pessoas da Adobe que escrevem sobre Flash e Flex.
- De uma olhada na [lista](#) que mantenho sobre frameworks, bibliotecas e outros recursos baseados no Flex.

E agora, eu acho, você pode entender porque quando um desenvolvedor Flex me pergunta o que é, eu olho fixamente para ele, e por um tempo, todas essas milhares de palavras que escrevi passam na frente dos meus olhos. Há tanta coisa pra falar, o que diabos eu poderia

dizem em uma linha!? Até agora, o que posso responder é: “Flex é incrível cara, confira nesse artigo”.

Se você tiver comentários, por favor deixe-o nesta página. Obrigado!