# 1月1号

上午去中国银行把存折挂了失，并马上不办了一个。中午与老婆的朋友的韩姐去吃了猪肚鸡。

今天研究了Tooltip、Popup、ComboBox的实现；

ComboBox的实现：

TextBox：显示当前选择的项的字符串表示，并支持Search；

ContentPresenter：显示当前选择项的内容；

ToggleButton：下拉按钮

Popup：显示集合项

从这个地方可以看出，Popup需要成为逻辑树的一部分，否则Popup中的 ItemsPresenter就无法获取到Selecor的Items。

从WPF的实现看，如果要实现AutoComplete的话，还是有一些距离。根据输入部分内容，查找数据库中符合条件的记录，然后把这些记录替换目前的列表。当然，如果已经选定了话，此时列表中，应该会只有一项。

需要接管TextBox的输入，然后查询，返回结果，到ComboBox，理论上应该是可以的吧。

# 1月2号

ComboBox的Popup是什么时候打开的，又是如何关闭的呢？

在ComboBox的模板中，把ToggleButton'的IsDropDownOpen绑定到了Popup的isOpen上，这样当IsDorpDownOpen发生变化后，就会打开Popup。

在Template上，ToggleButton的isCheck绑定到了ComboBox的IsDropDownOpen上。从而在这个按钮点击时保证下拉Popup。

PopupControlService：

注册输入事件侦听，来处理Tooltip问题；当鼠标悬停在一个元素上时，发出FindToolTip事件，这个事件是冒泡方式的，在元素树上向上追溯，直到有元素发现自己存在tooltip属性。

这个类本身是一个单例

ToolTip：

在FrameworkElement上，ToolTip作为一个属性存在的，因此ToolTip并不是逻辑树的一部分，虽然ToolTip有可能是一个控件。

在PopupControlService静态初始化中，注册了输入侦听，判断鼠标的MouseOver是否直接DirectOver，然后判断

ToolTip静态注册了FindToolTip事件侦听。只要有元素报告发现，然后创建ToolTip，这个ToolTip是全局唯一的，也就是说任何时候，都只有一个ToolTip。设置ToolTip的isOpen=true。然后设置定时器，到时候关闭这个ToolTip。

总结一下，首先判断DirectOver元素，然后通过FindTolTip事件来递归查找元素的ToolTip（这个事件是冒泡策略的），找到拥有ToolTip的元素，然后创建ToolTip，如果已经定义了ToolTip的话，就直接使用这个ToolTip。

问题是，设置了打开标志后，什么时候显示这个ToolTip呢？

在设置IsOpen=true后，会触发DP上的ToolTip.OnInOpenChanged（）在这个方法中，为ToolTip创建了Popup，从而显示出来ToolTip吗？

创建Popup后，设置Popup的内容，最后把ToolTip的IsOpen绑定到Popup的isOpen上，在Popup的isOpen属性变化中，为true，就显示Popup，否则就隐藏Popup，呵呵，总算是搞清楚Popup是如何显示的了！

# Popup实现

在WPF中，需要创建Window，然后把内容设置到这个Window中，在Summer中，这个与Window相关的处理就没有必要了。

一个是固定创建dom元素——div来包容Popup的内容，另一个是动态创建，不需要时销毁。

还需要研究WPF，Popup的window是如何管理的。

从其实现可以看出，需要的window是每一次都创建的，在popup关闭时销毁，呵呵，在windows中，window是资源，是要谨慎使用的。

在Summer中，这个window可以对应到div上，此时这个div是否需要动态管理呢？在打开时，创建div加入到dom，在close时，从document中移走呢。

还是固定一个div，在hide中，设置其display为none，在open时，通过绝对定位来显示这个div呢？

# Asp.net

应该是类似于jsf的一种技术，代码在服务器端执行，需要构造构建树。代码在元素树上执行。这个路子与目前Summer的思路区别太大。

# Dialog

在js中实现dialog的基本思路：创建一个div，设置这个div的zindex，然后在创建dialog需要的div，把对话框的内容渲染到这个div上，也就是追加到这个div的孩子上。

如果dialg中继续显示Dialog，这个过程是可以递归的。

分为模态和非模态两种，对于模态，就是对mask div的点击时，不关闭当前的dialog，非模态对话框时，点击mask div，就关闭当前的dialog。

这个Mask Div基于Document的body来创建。

# Window

在WPF中，window作为一个根元素，与Page并列。到了Summer中，这个Window就不能再享有Window的规格了，需要被极大的简化才行。

最终可以是借鉴Dialog的方式来进行。

# Calendar

# Grid

在这个布局中，允许多个控件堆叠在一起，看样子这个特性是需要的。

# CheckBox

在实现时，有一个问题，就是dom的堆叠，是否可以采用定位的方式，让一个单元格中的多个元素实现堆叠呢？

# Css的position

Static：就是默认的定位；

Relative：相对与正常位置的的偏移量；

Absolute：相对父级元素中第一个设置了非默认定义的；

Fixed；相对于浏览器窗口的；

# Js Drag Drop

研究一下js下的Drag and Drop

# 1月3号

几天计划：

1. 测试一下定位div，为popup做准备；
2. ~~研究一下js的timer，为tooltip做准备；~~
3. ~~实验css的position，解决grid的中单元格的元素堆叠；~~
4. ~~完成StackPanel；~~
5. ~~实验Binding中的ElmentName；~~

# Js Timer学习

理解这个timer的关键是明白，js是在单一线程中运行的；

当然，首先需要明白操作系统的一些机制。操作系统本质上来说，是事件驱动的。好比一个死循环，检测输入，然后执行输入导致的事件处理。Js中的代码就是通过js的事件来驱动的，如果没有事件发生，操作系统就处于等待状态。

Timer本质是一个定时事件，创建一个定时器，就是在调度队列中注册了一个处理，等待定时时间到达后执行。如果在一个事件处理中，由于这个事件处理耗费的时间，超过了定时间隔，对于timerOut来说，就会被延迟，对于internal来说，发生的多次，可能会保留一次了。

通过setInternal、setTimerOut来创建定时器，由于SetInternal创建的定时器需要被清除，因此setImternal会返回一个id，通过window.ClearInternal来取消这个定时器。否则这个定时器会一直执行下去的。

# Summer中的定时器

可以采取wpf的封装方式，通过new Timer的形式来创建，通过startTimer、StopTimer来启动和停止定时器。

经过一个多小时的调试，总算是基本上可用了。

这个地方有一个小心得需要记录下来。

在js中，事件的回掉是有浏览器管理的，对于dom元素的回掉，此时的this是dom元素，对于定时器的回掉，此时的this是window。因此如果在这个回掉方法中使用this，就需要明白此时这个this的含义，如果在要调用自己的this，此时就会出现问题了。

# Position

学习web编程这么多年，这是第一次实验css的position属性。顾名思义，这个属性就是放置、定位置的意思，也就是如何把这个元素显示在浏览器上。

浏览器默认的处理是文档流，也就是按照文档元素定义的顺序，分成两类元素来进行布局，碰到块元素，就会另起一行，碰到行元素就会挨着前一个元素来放置，当然行元素最终也是需要位于块元素中。

如果仅仅是这么简单，网页最终的网页会比现在的单调很多。

例如：有的网站在整个网页上游走的广告，文本环绕的图片，排成一行的导航条等等。

因此浏览器的设计者们引入一些允许改变默认特性的机制，当然无论是什么机制，基础首先是文档流的处理。

# CSS

可以用几个概念来深刻阐述：

1. 样式表就是一个规则集
2. 规则：选择器（作用的范围）、声明（Declaration，宣告、说明）

# Grid的单元格元素堆叠

经过实验，证明，如果是通过absolute的方式，确实是可以把元素进行堆叠的；在td中如果有多个元素的话，

# CheckBoxTest

在这个测试中，进一步又测试了同样的Binding，呵呵，效果很是不错的，就是有点慢。

# WrapPanel

几个属性

1. Orientation，方向，是水平流，还是垂直流；
2. ItemWidth
3. ItemHeight

第一个属性，在浏览器上不容易实现，可以忽略这个属性；

后面两个属性可以使用。

基本上实现。

# 1月4号

昨天还是没有搞明白这个ElementName作为Binding的Source的运行机制。今天无论如何要攻克这个难关。

# NameScope

按照WPF的实现，要求根元素上注册有NameScope的对象，来管理名称空间。在每一个通过Name、x：Name设置的Name，都需要注册自己到根元素的NameScope中。

但是Template和Style是如何做到的呢？

# Binding的ElementName

呵呵，终于把这个跑通了。给自己来个赞！但是还需要把代码运行路线梳理一下。现在的场景是，首先通过ElementName进行了binding，此时的结果显示出来是正确的，然后通过一个按钮修改ElementName对应元素的对应属性，此时的结果显示也是正确的。

# Binding的RelativeSource

使用这个数据源的Binding也跑通了。呵呵，还真是费了九牛二虎之力，不过也算是不虚此行吧。

这个关键是能够在元素树上找到匹配的元素，然后作为源，与ElementName之间后半段是一样的。

第二个关键是在AttachOverride部分，由于需要使用元素树作为上下文，因此Attach分成两个部分，第二部分，需要在布局完成后，在完整的元素树上进行，因此把后半段作为一个句柄添加到LayoutUpdated事件中，在Arrange时，触发这个事件从而完成完整的binding了。

# 任务

1. ~~测试DataProvider~~
2. 测试CollectionViewSource
3. 测试模板中的ElementName和RelativeSource
4. 完成TabControl
5. Command
6. 快捷键
7. Popup、ToolTip

# New apply

Js的new本质是，创建一个空对象，然后以这个对象为上下文调用构造函数，基于这个原理，当然自己可以通过apply来调用构造函数了。

呵呵，也是一个收获吧。

# 1月5号

接下来任务：

1. Binding的Converter和Exception处理；
2. Command
3. Popup、Tooltip

# ObjectDataProvider

这个总算是跑通了，两个方面，一个是使用构造函数的、一个是调用普通函数的。这个地方涉及到一个Converter的问题还没有解决好，需要进一步处理。

# 总结

有了ObjectDataProvider后，调用后台服务器上的数据成为了可能，将来通过编写ObjectDataProvider，在这个类中，调用服务器端的数据，渲染到界面上，否则界面再灵活也是瓷娃娃而已。

# MessageBox

需要这么一个类，来显示各种交互的提示信息。

可以认为以预先实现的一些Window而已；

# Window

这个需要实现，需要处理好与Popup之间的关系。在WPF中，Popup是与Window没有关系的，仅仅都需要Window的Hwnd而已。

实现Window后，可以利用Window来实现对话框，以及一般的弹出式窗口了。

可以在一个单独的文件中定义Window的界面和交互，然后Page的click中显示这个Window，当然，也可以直接在Page中定义Wndow类，然后使用，从而保证而之间的交互。

但是在XAML上，就不好处理。因为在WPF的设计中，这个Window也是一个根元素。这个根元素如果需要显示在Page中，就只能使用导航了。

但是在设计上，

先实现Popup也没有关系。

# Popup

在布局中，需要过滤掉（Popup本身已经把自己的Visibility，设置为固定的Collapse了），布局时，需要Arrange，但是不能把对应的Dom元素添加到文档中。

嗯，如果不扩展进来也不行，是要扩展进来，只是需要进行Position。这个Position可能要采用绝对定位。绝对定位后，这个DOM元素就已经脱离了正常文档流，其他元素可以当其不存在了。

这个时候，就可以通过显示和隐藏来实现Popup了。

# Tooltip

就是FrameworkElement的一个属性，这个属性的类型是Object，也就是允许是任何类型，当然一般是字符串和ToolTip类型了。

如果是String的话，最终需要使用模板，将其显示出来。

如果是ToolTip的话，当然也是需要模板的，只是此时的Tooltip是一个内容控件了。

需要在鼠标事件上，侦听mouseOver和MouseOut，MouseOver发生的话，判断是否有Tooltip，如果有，就设置定时器，准备显示，也就是把显示挂接到定时器中，如果发生了MouseOut，就清除定时器，或者隐藏已经显示的tooltip了。

这个tooltip最终是显示在popup中的。

嗯，这个地方有点复杂，可以考虑，引入一个选项，Simple、Complex来辅助解决，如果是Simple的话，就采用dom的提示，否则就采用复杂的提示。

# Mouse

这个地方还遗留了一些问题。

可以借鉴WPF的思路，在MouseDevice中创建一些侦听，分为Pre、Post两类，中间的应该就是用户的处理了，这些侦听就是系统使用的了。

这个地方的全局状态元素，需要进一步考察。例如，DirectOverElement、原来的MouseOver元素和当前的MouseOver元素。

在EventListenerManager中的注册，可以认为是提供个开发者的，

# SetCapture

这个方法，IE和FF都支持，只有Chrome不行，那就按照FF和IE的方式来处理吧。

# Mouse事件

MouseDown、MouseUp这两个事件，可以直接采用dom元素的事件来实现。不会有歧义。可以直接触发逻辑控件的事件；

MouseMove事件就有点讲究了，对于同一个逻辑控件的多个dom元素，其上发生的MouseMove需要被分析合并为对应的逻辑控件事件。当发生mouseMove时，是可以直接Raise这个逻辑控件的事件。

MouseOver和MouseMove的发生顺序。

怎么才注意，WPF中没有MouseOver事件吗？

MouseEnter、MouseLeave、MouseMove

MouseDown、MouseUp。

意识到一个问题，在WPF中，mouseEnter、mouseLeave是针对一个逻辑控件的，MouseEnter、MouseLeave都是direct策略的，这样到了Summer中，一种可能是一个控件由多个DOM组成，此时在dom上注册的鼠标进入和出来后，就有可能会比较复杂了。

目前的设计，一个控件至少对应一个dom，但是在dom元素中，多个dom其实都是对应一个控件，如果能够记录下来，这个问题应该不会引起麻烦。

这个地方的问题是，当在一个控件的孩子上进行鼠标移动时，也会导致父容器的mouseOut事件的产生，从而导致问题了。

分为两种情况：

1. 首先是一个逻辑控件，孩子节点是其组成部分，对使用者体现不出来这些孩子节点；此时由于每一个dom元素，其source都是同一个逻辑控件，如果加以判断，应该不会有问题；
2. 孩子节点时另外一个逻辑控件，此时mouseOver进入了孩子的话，容器就是应该不再是mouseOver了。

在MouseDevice中，针对mouseOver，记录其逻辑控件，在mouseOut中，并不需要进行处理。如果其他的dom发生了该事件，那么判断现在的逻辑控件和之前的是否一致即可。

如果mouseOver中，如果发生在一个新控件上，就设置这个控件的mouseOverProperty属性好了。

但是此时有另一个问题，就是dom事件来实现mouseEnter、mouseLeave这两个事件时，也是针对逻辑控件的，如果一个控件由多个dom元素构成，如果是简单的依赖dom元素事件的话，就会在同一个逻辑控件上触发多次mouseEnter、mouseLeave了。

呵呵，应该快明白了。

需要把dom元素的事件作为原始事件，在更高层次上进行分析合并，模拟出来逻辑控件需要的事件。

发生了mouseOver后，比较当前的mouseOver元素和发生事件的元素是否一致，如果一致，此时什么也不需要做，如果不一致，就需要对新的控件发生mouseOver了。

在dom中，mouseOut和mouseOver的发生顺序应该是mouseOut发生，然后mouseOver发生。如果这样的话，问题就变得复杂了。

实验了一下，确实是这样，mouseOut先发生，mouseOver才发生。

经过一番绞尽脑汁，想出了如下的解决方案：

对于MouseLeave，进行如下的假设，控件元素一定会包含着构成自己的直接dom孩子。只要源dom==事件发生的dom，才说明是离开，否则就认为是内部的移动，并不需要采取冒泡的处理。

MouseOver也是，第一次发生后，判断后面发生的是否是在同一个逻辑控件内，如果是在同一个，就不触发WPF事件。

Preview系列的鼠标事件也是这个策略。

# 鼠标事件实现

在MouseDevice中，设置统一的类来处理，这个单例类需要记录全局状态。

在MouseOver中记录发生事件的逻辑控件，在MouseOut中清除这个记录。需要照顾处理好IsMouseOver属性的设置。

在处于Capture状态下时，鼠标的mouseOver应该不要发生了吧。

在设置Capture时，需要设置一个CaptureMode标志，此时其他元素就不要在发生鼠标事件了吧。

# 键盘事件

相对鼠标事件的问题，键盘事件面临着同样的问题。

需要对组成控件的多个dom元素的键盘事件进行综合、分析，解析出来对应的控件键盘事件。

这个地方相对于鼠标应该会简单一些吧。

# 1月6号

今天把Window的代码迁移一下。呵呵这个类真够庞大的，代码差不多8千多行。

可能需要一上午才行，不过绝大多数的代码是要删除的。可以先采取注释掉的方式吧。

# 窗口处理

按照WPF的规则，窗口是需要单独的XAML文件来描述。在Summer中，这个应该是没有问题的。

每一个独立的XAML文件，等价于一个类，类名称就是通过x:class指定的属性。因此，窗口自然是可以作为一个类引入到page中来进行使用，自然也就没有问题。

# 忽然想到

ContentMenu是什么呢？

难道也是Popup吗？赶快看看吧。

# Popup

这个类到了进行化简得时候了。

# 1月7号

看看ContextMenu的代码，及其实现机制。

ContextMenu确实是通过Popup来实现的。需要时候，通过HookParentPopup（）方法，来创建自己的Popup，从而已弹出窗口的形式来显示。

可以看出，Popup是WPF的一个基础设施，很多控件都利用了Popup，需要攻克这个拦路虎才行。

对于Popup的动画，可以认为是作用在div上的，只要设置一个动画css即可，此时应该就出现动画，动画完成要清除吗，如何再次启动动画。应该是设置css。

# 动画

在WPF中，通过DP的基础设施来自己实现了动画，但是到了浏览器上，由于css3支持动画，此时这个动画，对用户来说是透明的，也就是说，并不在需要在DP上提供支持。

只需要设置对应的css即可。因此需要保留Transform部分的代码，以便可以利用css3的变换了。

# Popup Placement Behaviour

A [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) control displays content in a separate window that floats over an application. You can specify the position of a [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) relative to a control, the mouse, or the screen by using the [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx), [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx), [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx), [HorizontalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.horizontaloffset(v=vs.110).aspx), and [VerticalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.verticaloffset(v=vs.110).aspx) properties. These properties work together to give you flexibility in specifying the position of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx).

一个Popup控件用来在独立窗口区域显示内容，漂浮在应用上。可以通过PlacementTarget、Placement、PlacementRectangle、HorizontalOffset、VerticalOffset这个5个属性来设置相对于Control、mouse、Screen的位置。这些属性组合到一起提供了灵活的定位。

<Canvas Width="200" Height="150">

<Image Name="image1"

Canvas.Left="75"

Source="Water\_lilies.jpg" Height="200" Width="200"/>

<Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"

Placement="Bottom">

<TextBlock FontSize="14" Background="LightGreen">Placement=Bottom</TextBlock>

</Popup>

<Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"

Placement="Top">

<TextBlock FontSize="14" Background="LightGreen">Placement=Top</TextBlock>

</Popup>

<Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"

Placement="Left">

<TextBlock FontSize="14" Background="LightGreen">Placement=Left</TextBlock>

</Popup>

<Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"

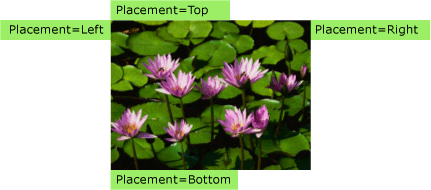
Placement="Right">

<TextBlock FontSize="14" Background="LightGreen">Placement=Right</TextBlock>

</Popup>

</Canvas>

上述大脚本将产生如下的效果：



[Definitions of Terms: The Anatomy of a Popup](javascript:void(0))

The following terms are useful in understanding how the [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx), [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx), [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx), [HorizontalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.horizontaloffset(v=vs.110).aspx), and [VerticalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.verticaloffset(v=vs.110).aspx) properties relate to each other and the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx):

* Target object
* Target area
* Target origin
* Popup alignment point

These terms provide a convenient way to refer to various aspects of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) and the control that it is associated with.

Target Object

The *target object* is the element that the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is associated with. If the [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) property is set, it specifies the target object. If [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) is not set, and the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) has a parent, the parent is the target object. If there is no[PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) value and no parent, there is no target object, and the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is positioned relative to the screen.

The following example creates a [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) that is the child of a [Canvas](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.canvas(v=vs.110).aspx). The example does not set the [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) property on the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). The default value for [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [PlacementMode.Bottom](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), so the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) appears below the [Canvas](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.canvas(v=vs.110).aspx).

<Canvas Margin="5" Background="Red" Width="200" Height="150" >

<Ellipse Canvas.Top="60" Canvas.Left="50"

Height="85" Width="60"

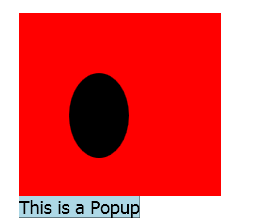
Fill="Black"/>

<Popup IsOpen="True" >

<TextBlock Background="LightBlue" FontSize="18">This is a Popup</TextBlock>

</Popup>

</Canvas>



he following example creates a [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) that is the child of a [Canvas](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.canvas(v=vs.110).aspx), but this time the [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) is set to ellipse1, so the popup appears below the [Ellipse](http://msdn.microsoft.com/zh-cn/library/system.windows.shapes.ellipse(v=vs.110).aspx).

<Canvas Margin="5" Background="Red" Width="200" Height="150" >

<Ellipse Name="ellipse1"

Canvas.Top="60" Canvas.Left="50"

Height="85" Width="60"

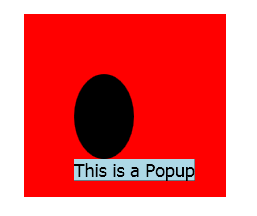
Fill="Black"/>

<Popup IsOpen="True" PlacementTarget="{Binding ElementName=ellipse1}">

<TextBlock Background="LightBlue" FontSize="18">This is a Popup</TextBlock>

</Popup>

</Canvas>



For [ToolTip](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.tooltip(v=vs.110).aspx), the default value of [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [Mouse](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx). For [ContextMenu](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.contextmenu(v=vs.110).aspx), the default value of [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [MousePoint](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx). These values are explained later, in "How the Properties Work Together."

Target Area

The *target area* is the area on the screen that the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is relative to. In the previous examples, the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is aligned with the bounds of the target object, but in some cases, the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is aligned to other bounds, even if the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) has a target object. If the [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) property is set, the target area is different than the bounds of the target object.

The following example creates two [Canvas](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.canvas(v=vs.110).aspx) objects, each one containing a [Rectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.shapes.rectangle(v=vs.110).aspx) and a [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). In both cases, the target object for the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is the [Canvas](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.canvas(v=vs.110).aspx). The [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) in the first [Canvas](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.canvas(v=vs.110).aspx) has the [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) set, with its [X](http://msdn.microsoft.com/zh-cn/library/system.windows.rect.x(v=vs.110).aspx), [Y](http://msdn.microsoft.com/zh-cn/library/system.windows.rect.y(v=vs.110).aspx), [Width](http://msdn.microsoft.com/zh-cn/library/system.windows.rect.width(v=vs.110).aspx), and [Height](http://msdn.microsoft.com/zh-cn/library/system.windows.rect.height(v=vs.110).aspx)properties set to 50, 50, 50, and 100, respectively. The [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) in the second [Canvas](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.canvas(v=vs.110).aspx) does not have the [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) set. As a result, the first [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is positioned below the [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) and the second [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is positioned below the [Canvas](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.canvas(v=vs.110).aspx). Each [Canvas](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.canvas(v=vs.110).aspx) also contains a [Rectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.shapes.rectangle(v=vs.110).aspx) that has the same bounds as the [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) for the first [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). Note that the [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) does not create a visible element in the application; the example creates a [Rectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.shapes.rectangle(v=vs.110).aspx) to represent the[PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx).

<StackPanel Orientation="Horizontal" Margin="50,50,0,0">

<Canvas Width="200" Height="200" Background="Red">

<Rectangle Canvas.Top="50" Canvas.Left="50"

Width="50" Height="100"

Stroke="White" StrokeThickness="3"/>

<Popup IsOpen="True" PlacementRectangle="50,50,50,100">

<TextBlock FontSize="14" Background="Yellow"

Width="140" TextWrapping="Wrap">

This is a popup with a PlacementRectangle.

</TextBlock>

</Popup>

</Canvas>

<Canvas Width="200" Height="200" Background="Red" Margin="30,0,0,0">

<Rectangle Canvas.Top="50" Canvas.Left="50"

Width="50" Height="100"

Stroke="White" StrokeThickness="3"/>

<Popup IsOpen="True">

<TextBlock FontSize="14" Background="Yellow"

Width="140" TextWrapping="Wrap">

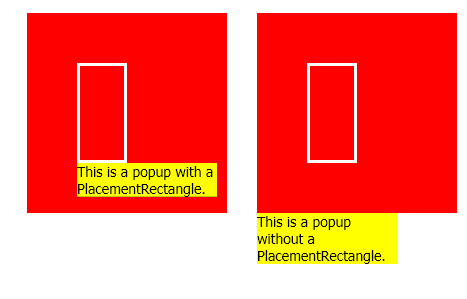
This is a popup without a PlacementRectangle.

</TextBlock>

</Popup>

</Canvas>

</StackPanel>



Target Origin and Popup Alignment Point

The *target origin* and *popup alignment point* are reference points on the target area and popup, respectively, that are used for positioning. You can use the [HorizontalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.horizontaloffset(v=vs.110).aspx) and [VerticalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.verticaloffset(v=vs.110).aspx) properties to offset the popup from the target area. The[HorizontalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.horizontaloffset(v=vs.110).aspx) and [VerticalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.verticaloffset(v=vs.110).aspx) are relative to the target origin and the popup alignment point. The value of the [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) property determines where the target origin and popup alignment point are located.

The following example creates a [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) and sets the [HorizontalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.horizontaloffset(v=vs.110).aspx) and [VerticalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.verticaloffset(v=vs.110).aspx) properties to 20. The [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) property is set to [Bottom](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) (the default), so the target origin is the bottom-left corner of the target area and the popup alignment point is the top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx).

<Canvas Width="200" Height="200" Background="Yellow" Margin="20">

<Popup IsOpen="True" Placement="Bottom"

HorizontalOffset="20" VerticalOffset="20">

<TextBlock FontSize="14" Background="#42F3FD">

This is a popup.

</TextBlock>

</Popup>

</Canvas>

The following illustration shows the result of the preceding example.

Popup with HorizontalOffset and VerticalOffset



[How the Properties Work Together](javascript:void(0))

The values of [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx), [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx), and [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) need to be considered together to figure out the correct target area, target origin, and popup alignment point. For example, if the value of [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [Mouse](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), there is no target object, the [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is ignored, and the target area is the bounds of the mouse pointer. On the other hand, if [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [Bottom](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), the [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) or parent determines the target object and [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) determines the target area.

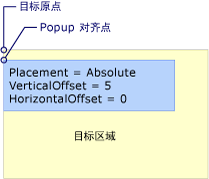
The following table describes the target object, target area, target origin, and popup alignment point and indicates whether [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) and [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) are used for each [PlacementMode](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) enumeration value.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PlacementMode** | **Target object** | **Target area** | **Target origin** | **Popup alignment point** |
| [Absolute](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Not applicable. [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) is ignored. | The screen, or [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) if it is set. The [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is relative to the screen. | The top-left corner of the target area. | The top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [AbsolutePoint](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Not applicable. [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) is ignored. | The screen, or [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) if it is set. The [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is relative to the screen. | The top-left corner of the target area. | The top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [Bottom](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) or parent. | The target object, or [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) if it is set. The [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is relative to the target object. | The bottom-left corner of the target area. | The top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [Center](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) or parent. | The target object, or [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) if it is set. The [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is relative to the target object. | The center of the target area. | The center of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [Custom](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) or parent. | The target object, or [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) if it is set. The [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is relative to the target object. | Defined by the[CustomPopupPlacementCallback](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.custompopupplacementcallback(v=vs.110).aspx). | Defined by the[CustomPopupPlacementCallback](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.custompopupplacementcallback(v=vs.110).aspx). |
| [Left](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) or parent. | The target object, or [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) if it is set. The [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is relative to the target object. | The top-left corner of the target area. | The top-right corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [Mouse](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Not applicable. [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) is ignored. | The bounds of the mouse pointer. [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is ignored. | The bottom-left corner of the target area. | The top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [MousePoint](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Not applicable. [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) is ignored. | The bounds of the mouse pointer. [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is ignored. | The top-left corner of the target area. | The top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [Relative](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) or parent. | The target object, or [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) if it is set. The [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is relative to the target object. | The top-left corner of the target area. | The top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [RelativePoint](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) or parent. | The target object, or [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) if it is set. The [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is relative to the target object. | The top-left corner of the target area. | The top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [Right](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) or parent. | The target object, or [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) if it is set. The [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is relative to the target object. | The top-right corner of the target area. | The top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [Top](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | [PlacementTarget](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementtarget(v=vs.110).aspx) or parent. | The target object, or [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) if it is set. The [PlacementRectangle](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placementrectangle(v=vs.110).aspx) is relative to the target object. | The top-left corner of the target area. | The bottom-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |

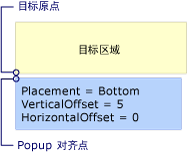
The following illustrations show the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx), target area, target origin, and popup alignment point for each [PlacementMode](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) value. In each figure, the target area is yellow, and the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is blue.

Placement is Absolute or AbsolutePoint

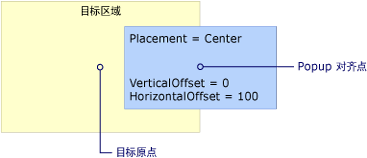
Placement is Bottom



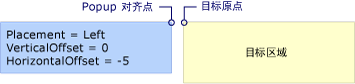
Placement is Center



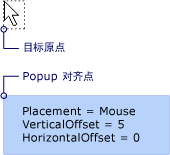
Placement is Left



Placement is Mouse



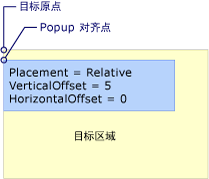
Placement is MousePoint



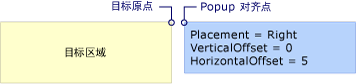
Placement is Relative or RelativePoint



Placement is Right



Placement is Top



[When the Popup Encounters the Edge of the Screen](javascript:void(0))

For security reasons, a [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) cannot be hidden by the edge of a screen. One of the following three things happens when the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) encounters a screen edge:

The popup realigns itself along the screen edge that would obscure the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx).

The popup uses a different popup alignment point.

The popup uses a different target origin and popup alignment point.

These options are described further later in this section.

The behavior of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) when it encounters a screen edge depends on the value of the [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) property and which screen edge the popup encounters. The following table summarizes the behavior when the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) encounters a screen edge for each [PlacementMode](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) value.

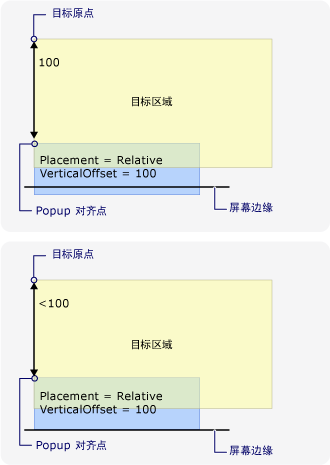
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PlacementMode** | **Top edge** | **Bottom edge** | **Left edge** | **Right edge** |
| [Absolute](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Aligns to the top edge. | Aligns to the bottom edge. | Aligns to the left edge. | Aligns to the right edge. |
| [AbsolutePoint](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Aligns to the top edge. | The popup alignment point changes to the bottom-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). | Aligns to the left edge. | The popup alignment point changes to the top-right corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [Bottom](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Aligns to the top edge. | The target origin changes to the top-left corner of the target area and the popup alignment point changes to the bottom-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). | Aligns to the left edge. | Aligns to the right edge. |
| [Center](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Aligns to the top edge. | Aligns to the bottom edge. | Aligns to the left edge. | Aligns to the right edge. |
| [Left](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Aligns to the top edge. | Aligns to the bottom edge. | The target origin changes to the top-right corner of the target area and the popup alignment point changes to the top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). | Aligns to the right edge. |
| [Mouse](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Aligns to the top edge. | The target origin changes to the top-left corner of the target area (the bounds of the mouse pointer) and the popup alignment point changes to the bottom-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). | Aligns to the left edge. | Aligns to the right edge. |
| [MousePoint](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Aligns to the top edge. | The popup alignment point changes to the bottom-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). | Aligns to the left edge. | The popup alignment point changes to the top-right corner of the popup. |
| [Relative](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Aligns to the top edge. | Aligns to the bottom edge. | Aligns to the left edge. | Aligns to the right edge. |
| [RelativePoint](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Aligns to the top edge. | The popup alignment point changes to the bottom-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). | Aligns to the left edge. | The popup alignment point changes to the top-right corner of the popup. |
| [Right](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | Aligns to the top edge. | Aligns to the bottom edge. | Aligns to the left edge. | The target origin changes to the top-left corner of the target area and the popup alignment point changes to the top-right corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). |
| [Top](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) | The target origin changes to the bottom-left corner of the target area and the popup alignment point changes to the top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). In effect, this is the same as when [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is[Bottom](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx). | Aligns to the bottom edge. | Aligns to the left edge. | Aligns to the right edge. |

Aligning to the Screen Edge



A [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) can align to the edge of the screen by repositioning itself so the entire [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is visible on the screen. When this occurs, the distance between the target origin and popup alignment point might differ from the values of [HorizontalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.horizontaloffset(v=vs.110).aspx) and[VerticalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.verticaloffset(v=vs.110).aspx). When [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [Absolute](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), [Center](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), or [Relative](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) aligns itself to every screen edge. For example, assume that a [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) has [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) set to [Relative](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) and [VerticalOffset](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.verticaloffset(v=vs.110).aspx) set to 100. If the bottom edge of the screen hides all or part of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx), the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) repositions itself along the bottom edge of the screen and the vertical distance between the target origin and popup alignment point is less than 100. The following illustration demonstrates this.

Popup aligns to the edge of the screen



Changing the Popup Alignment Point

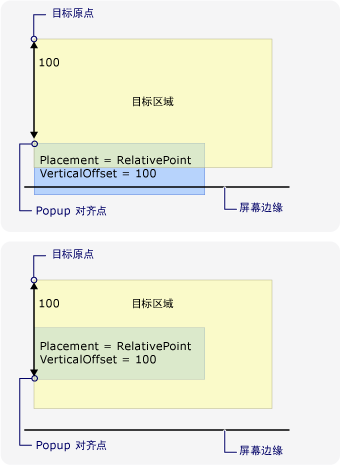


If [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [AbsolutePoint](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), [RelativePoint](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), or [MousePoint](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), the popup alignment point changes when the popup encounters the bottom or right screen edge.

The following illustration demonstrates that when the bottom screen edge hides all or part of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx), the popup alignment point is the bottom-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx).

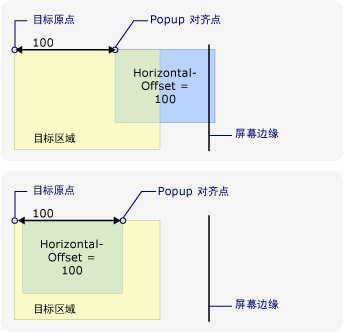
Popup encounters bottom edge of the screen and changes the popup alignment point

The following illustration demonstrates that when the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is hidden by the right screen edge, the popup alignment point is the top-right corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx).



Popup encounters right edge of the screen and changes the popup alignment point

If the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) encounters the bottom and right screen edges, the popup alignment point is the bottom-right corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx).



Changing the Target Origin and Popup Alignment Point

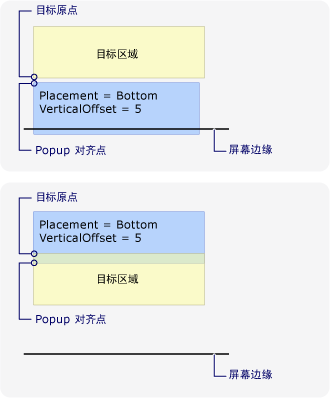


When [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [Bottom](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), [Left](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), [Mouse](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), [Right](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), or [Top](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx), the target origin and popup alignment point change if a certain screen edge is encountered. The screen edge that causes the position to change depends on the [PlacementMode](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) value.

The following illustration demonstrates that when [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [Bottom](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) and the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) encounters the bottom screen edge, the target origin is the top-left corner of the target area and the popup alignment point is the bottom-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx).

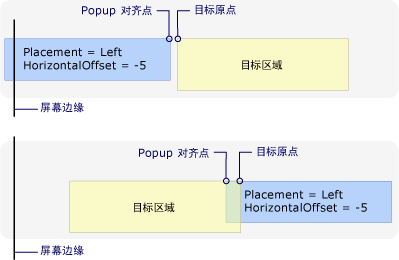
Placement is Bottom and the popup encounters the bottom edge of the screen

The following illustration demonstrates that when [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [Left](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) and the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) encounters the left screen edge, the target origin is the top-right corner of the target area and the popup alignment point is the top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx).



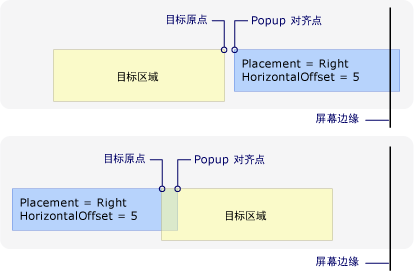
Placement is Left and the popup encounters the left edge of the screen

The following illustration demonstrates that when [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [Right](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) and the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) encounters the right screen edge, the target origin is the top-left corner of the target area and the popup alignment point is the top-right corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx).



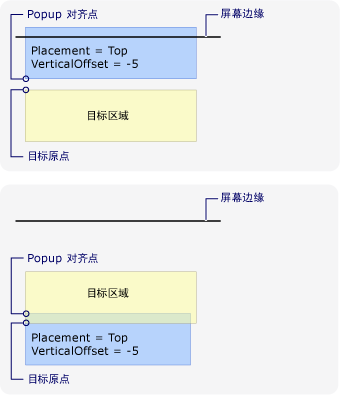
Placement is Right and the popup encounters the right edge of the screen

The following illustration demonstrates that when [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [Top](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) and the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) encounters the top screen edge, the target origin is the bottom-left corner of the target area and the popup alignment point is the top-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx).

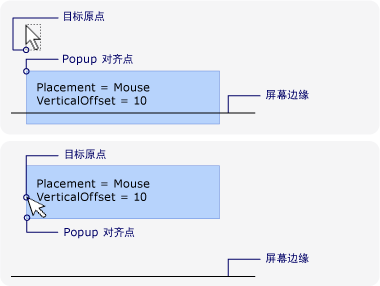


Placement is Top and the popup encounters the top edge of the screen

The following illustration demonstrates that when [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) is [Mouse](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx) and the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) encounters the bottom screen edge, the target origin is the top-left corner of the target area (the bounds of the mouse pointer) and the popup alignment point is the bottom-left corner of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx).



Placement is Mouse and the popup encounters the bottom edge of the screen



Customizing Popup Placement



You can customize the target origin and popup alignment point by setting the [Placement](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup.placement(v=vs.110).aspx) property to [Custom](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.placementmode(v=vs.110).aspx). Then define a [CustomPopupPlacementCallback](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.custompopupplacementcallback(v=vs.110).aspx) delegate that returns a set of possible placement points and primary axes (in order of preference) for the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx). The point that shows the largest portion of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is selected. The position of the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is automatically adjusted if the [Popup](http://msdn.microsoft.com/zh-cn/library/system.windows.controls.primitives.popup(v=vs.110).aspx) is hidden by the edge of the screen. For an example, see [How to: Specify a Custom Popup Position](http://msdn.microsoft.com/zh-cn/library/ms743607(v=vs.110).aspx).

# ToolTip

Tooltip的机制类似于ContextMenu，只是一个鼠标位置，一个是Target的底部。一个需要鼠标点击显示，一个是定时器显示而已吧。

首先，tooltip在XAML中定义后，其逻辑父亲是确定的。

到了该显示的时候，就创建一个Popup，然后交给Popup来显示就好了。

一个问题？此时Tooltip的Visual父亲会是谁呢？从代码看，创建好Popup后，就把Tooltip的属性binding到了Popup上。

# Popup

创建一个Hwnd对应的窗口。

创建一个PopupRoot作为Popup的孩子。这个是干什么用呢？

定义的内容是在Popup中进行Measure和Arrange呢还是在原来的逻辑容器上进行Measure和Arrange呢？

从StackPanel的代码看，如果是Collapse的话，在Measure中，或剔除这个控件。在Arrange中这个控件也会被踢出。

在Popup中，有一个RootPopup的元素，是一个Decorator，最终会把自己的Child设置到RootPopup中，作为其可视化的孩子。而RootPopup会被设置到HwndSource上，从而由这个HwndSource来调用其Measure和Arrange吧。

# Popup

从FrameworkElement元素继承实现了IAddChild接口。

但是其Child属性区别于Border（Decorator的子类）控件的Child属性。在Decorator中，设置Child的话，会同时维护到逻辑树和Visual中，在Popup中，仅仅是记录了这个属性，是一个依赖属性。在Decorator中，就不是一个DP了——不能进行Binding。

顺便回顾一下ContentControl，在这个控件中，关键属性是Content，这是一个DP，支持对内容的binding。这个Content如果是UI，就直接显示出来，当然仍然需要Template来支持，Visual Tree的连接是通过TemplatedParent来实现的。如果是UI的话，逻辑树和视觉树是不同的了。

Popup的Visibility总是Collapse，因此会不参与正常的layout流程。但是又需要显示出来，于是WPF的作者们，就引入了一个所谓的RootPopup，这个RootPopup作为Window的rootVisual，最终显示出来。

基于DOM的实现，去掉RootPopup，在Popup的ShowWindow中，判断是否已经创建了DOM，如果创建的话，先设置绝对定位，然后把DOM元素添加到自己的TargetElement中，然后设置display属性从而显示出来。

在Popup的ShowWindow中执行孩子的Arrange，和Measure，从而实现内容填充。如果此时是Tooltip的话，就执行Tooltip的Measure之类，从而应用模板，把自己显示出来。嗯，这个地方调用本身的Measure和Arrange不行？可以，只是作为孩子的时候，被排除到了布局之外。

一个难点是，如何实现WPF设置的Position机制，这个多个组合，在css中是否可以实现呢？

如果是鼠标的话，就获取当前的鼠标位置，嗯，这个地方需要总是接管每一个 dom元素的mouseMove，从而把鼠标位置记录下来，从这个角度看，还是需要集中的事件处理，这可能就是InputManager的设计初衷吧。对每一个事件都进行Pre、Post的处理，从而获取集中处理机制。

如果存在Target，就获取Target的dom的位置，然后计算出来popup的显示位置；

如果是PlacementRectangle的话，就参照这个矩形进行计算；

# Menu

在MenuBase中注册了MenuItem定义的PreviewClickEvent，而在MenuItem中使用了MenuBase，由于MenuBase是在静态初始化时使用了MenuItem，因此构成了循环参照。这个是循环参照的另一个情形，需要破解。

一个解决方案是，在MenuItem中注册MenuBase需要的事件处理，也就是把这句话放到MenuItem的静态初始化中。这个地方隐含着一个假定，就是Menu一定需要MenuItem才行。

# 明日任务

1. ~~解决字体相关的处理,SystemFonts，需要对这个类进行大改，可以先都设置为Arial字体。保留全局变量名称；~~
2. ~~解决颜色相关的处理~~

# 1月8号

字体部分应该是基本上ok了；但是这个地方的代码有些混乱，需要进一步处理才行。

呵呵，就是简单地设置Style，居然也需要花费大半个小时。

今天把，Calendar相关的几个类进行了迁移。

# 1月9号

今天继续Calendar，争取出来一些结果。

# DateTime

在c#中，采用的纳秒数，js中采用的是毫秒数。

是否可以修改DateTIme采用毫秒计数，这样的话，就不必担心其溢出了。

# Popup

今天还是现初步让Popup显示出来吧。

# 进入深水区了

当我们添加一个逻辑孩子时，由于其父亲的改变，而导致可能的样式更新，样式更新会更新样式定义的属性，如果定义了Template的话，此时模板也会得到更新，从而也就是用模板了。

但是Tooltip的话，如果在FrameworkElement中定义了Tooltip属性，到了MouseMove事件中显示这个Tooltip，此时是创建了一个Tooltip，此时ToolTIp的模板如何获得呢？

因为这个ToolTip并没有被添加任何逻辑父亲上，如何获取资源中定义的属性设置，尤其是ControlTemplate呢？

原来在Popup中采取了一种特殊机制，就是通过把自己注册到PlacementTarget上，然后采用强制更新孩子属性的方式，从而让Popup的Child可以使用资源中定义的模板了。

# Tooltip出来了

折腾了一番，Tooltip居然出来了。让子弹飞一会儿！

# 1月10号

天天22个月了。

今天坐高铁回西安。

4点多就起来了。

# MouseLeave

在物理DOM上，判断是否超出了逻辑控件的范围，如果超出，就可以发出MouseLeave了。呵呵，还是不行，因为如果超出的话，逻辑控件已经被更换了。

嗯，这个地方隐藏着一个机制，当mouseOut时，其实此时还附加了一个鼠标坐标，也就是发生mouseOut时的鼠标坐标，此时可以判断逻辑控件本身的dom的rectangle是否包含这个坐标，如果已经不包含了，就认为是超出了当前的逻辑控件了。可以发出mouseLeave了。

# MouseEnter

发生mouseOver时，判断当前的逻辑控件是否与之前的逻辑控件一致，在mouseout中，可能已经被清除了，此时要设置新的directOver了。并且发出mouseEnter事件。

# MouseMove

注册到dom上的侦听器，针对每一次的鼠标移动，给dom关联的逻辑控件发送mouseMove事件

# MouseUp、MouseDown

这两个事件的处理与mouseMove一样；

# Capture

在mouseDown中进行Capture，在mouseUp中释放，此时假设mousemove都会发送事件到这个侦听器上。

# Focus

可以在dom的focus事件中，进行处理，判断是否与此前的focus一致，如果不一致，就意味着发生了blur和focus了。当keyboard中维持住当前的focusElement。

# 任务

* Focus

接下来需要花费时间彻底研究一下focus的导航问题了。因为可以获取到dom元素的位置矩形，因此KeyboardNavigation的有些逻辑应该是可行的了。

* binding中的UpdateSourceTrigger

需要看看lostFocus这个机制该如何实现呢？

* Tooltip部分的Timer的逻辑?

这个地方的mouseMove事件，不要再是原始事件了吧，可以搞一个所有的事件在一起的PostProcess了，来作为侦听器，这样可能就会借鉴WPF的处理了。

# TabControl

这个控件的实现原理

TabControl的页签是一个Items容器，只是这个容器仅渲染Header而已（TabItem是一个HeaderedControl），接着是一个ContentPresenter，其内容binding到了SelectedContent上了。

这个地方的理解是，Items中的其他内容运行时并没有控件与其对应，除了当前选择的item。

关键是：

1. 页签部分的选择变化，视觉上的，然后设置SelectedContent，进入导致ContentPresenter被渲染。

# 回到了西安

# ReverseInheritProperty

这个机制需要花些时间才行。

UIElement类有两个属性：

IsMouseOver，确定是否鼠标在一个元素上，或者在它的一个子元素上。

IsMouseDirectlyOver，确定是否鼠标在一个元素上，但不在它的子元素上。

看样子IsMouseOver和IsMouseDirectlyOver是需要深刻理解的。

IsMouseOver是体现在元素树上的，只要孩子是MouseOver，其父元素都是MouseOver的。

而IsMouseDirectlyOver意思是只有一个元素。

IsMouseOver是一类属性的代表。这一类属性有如下的特点：

1. 是只读属性；
2. 没有保存到EffectiveValues中；
3. 虽然是只读属性，只是针对某一个层面来说的，对有些层面来说，还是可以写的。也就是总是需要设置这个属性的值，一般来说，叫做系统设置，用户是不能设置的；
4. 也是依赖属性，本质上还是需要具有DP的特点；这个地方一个问题，就是属性变化侦听如何解决呢？

原来是保存在了两个地方，一个是保存到EffectiveValues中，一个是保存在标志位上。

当我们定义了Trigger，针对IsMouseOver属性的，判断如果为true，就设置背景色了；但是是如何触发Trigger的呢？

既然是DP，那么就一定在UpdateEffectiveEntry上，判断是否发生了改变，如果发生了改变的话，就针对Dependent上的属性，进行Invalidate，在dp更新过程中，就会针对Trigger进行计算，从而让这个Trigger产生作用。

然后针对这个属性其实有两个事件，一个是MouseEnter、一个是MouseLeave；WPF也正是在判断MouseOver的过程中裁定这两个事件的。

也就是说，IsMouseOver属性要先发生，然后才是MouseOver了。

# 1月11号

一个调用顺序

当在FrameworkElement添加了孩子后，AddLogicalChild后，——FrameworkObject的ChangeLogicalParent——进入FrameworkElement的ChangeLogicalParent——进入FrameworkELemnt的OnNewParent——进入UIElement的SynchronizeReverseInheritPropertyFlags，在这个方法中维护——进入MouseDevice的ReEvaluateMouseOver，在这个方法中，进行了一项工作，就是设置了MouseOverTreeState 然后注册了一个回调，进入ReevaluateMouseOverAsync 来同步鼠标状态。

嗯，这个机制在js下有无必要呢？

大概的意思是，如果添加了一个孩子，这个孩子的MouseOver被设置了，此时应该需要把相应的属性设置到新的父亲上，本来是可以马上设置的，但是被延迟到了布局完成之后。

在MouseDevice中定义了一个DeferredElementTreeState类型的属性，是全局共享的，因为MouseDevice是作为一个单例而存在的。InputManager是一个单例，维护了一个MouseDevice的唯一实例。

假设在MouseOver事件中，改变了当前元素的孩子的话，应该是可以直接改变父亲元素的属性吗，但是由于此时缺少机制来改变父元素的这个属性了。因为这类属性都是通过另外的机制来实现的。

# InheritanceContext

这个机制还需要花些时间再深入一下。

先看看一个大拿写的关于InherianceContent的背景知识。

为什么需要继承上下文呢？

在元素树上，元素可以通过逻辑父亲和Visual父亲来获取父元素设置的继承属性——如果自己没有设置的话。

这个机制是建立在逻辑树和VisualTree上的。

例如

<Button>

<Button.Background>

<SolidBrush>

<SolodBrush>

</Button.Background>

</Button>

此时，这个SolidBrush的不是逻辑树和VisualTree的组成部分，也就不能从Button上获取继承属性；

但是有些时候，需要一个继承上下文，甚至是附加属性，看下面的例子

<Button>

<Button.Background>

<SolidBrush Color={Binding}>

<SolodBrush>

</Button.Background>

</Button>

此时这个Color就需要从DataContext上获取了。

为么满足这个需求，需要扩充继承机制，姑且称之为继承2.0吧。

{

如何实现这个机制呢？

增加一个InheritanceContext属性到DependencyObject上，记录这种特殊的情形，如何维护呢？当添加到EffectiveValues中时——当维护DP时，在PropertyChanged中，判断如果是Freezable类型或者说需要InheritanceContext的话，就把当前的DO添加到这个属性的继承上下文中。

扒开代码验证一下。呵呵，自己想的有点小家子气了，WPF的专家更加大手笔，在UpdateEffectiveValue中，为每一个对象都调用ProviderSelfAsInheritanceContext（）方法，在这个方法中，进行判断，如果是DependencyObject的话，就通过ProviderSelfAsInheritanceContext（）方法，把自己添加到Value的继承上下文中，调用AddInheritanceContext（）方法，当然这个指令是公开发给所有人的，但是只有有心人才会响应这个指令。

}

继续回到前文，从WPF实现看到，如下的几个类重载了AddInheritanceContext方法：

Freezable

FrameworkElement

FrameworkContentElement

TriggerBase

TriggerAction

BindingGroup

ViewCollectionSource

# MVVM

花时间又把MVVM录像看了一下。

呵呵，常看常新，本质是把与界面的交互变成ViewModel本身的操作；

数据属性、命令属性两个核心物件，分别binding到View的数据控件和Command控件上；

# MouseDevice

这个地方的一个代码是关于DeferredElementTreeState，在添加孩子到元素树上时，需要把孩子的属性反映到父元素的属性上。这个地方采用么延迟的方式，为什么要采取这个机制呢？担心元素树还没有构造好吗？

抛开这个插曲，剩下的代码就比较容易了。当一个元素发生了MouseOver时，除了设置自身的IsMouseOver属性，同时把这个属性向父元素传递，保证父元素的这个属性也同样设置。

WPF是在HitTest中检测到一个元素拥有的鼠标，于是就设置MouseOver的。

当然需要在MouseDevice中维持DirectlyMouseOver的元素。也需要维持MouseOver元素记录；

其实在Mouse'Device中只是记录了MouseOver一个元素，这个元素就是DirectlyMouseOver的元素。当然也同时设置了DirectlyMouseOver属性了。

对于MouseOver从代码看出来，虽然IsMouseOver会从当前元素向父祖先元素传播，但是MouseEnter、MouseLeave事件却只是发生在直接作用的元素上的。父元素的IsMouseOver属性虽然改变，却不会产生MouseEnter、MouseLeave事件。

嗯，这么处理有道理，虽然当前元素的MouseOver发生了变化，但是并不意味着父元素的发生MouseEnter、MouseLeave事件。

按照WPF的实现，只采用MouseOver事件即可。

或者只是判断MouseMove即可。

哪一个元素发生了MouseMove，检查是否当前的dom元素的source与MouseDevice中的mouseOver元素一致，如果一致，就不需要进行处理，否则就需要重新设置mouseOver，然后触发mouseEnter、mouseLeave事件，当然需要按照WPF的规范，对于IsMouseOver属性的维护（包括向父元素的传播）。

如果利用moueOver的话，也是可以的吧。只是有一个，那就是如果page上面的元素没有了mouseOver的话，就不会发生之前元素的mouseLeave了。

而如果采取mouseMove的话，可能还好一些。

MouseOut先发生、mouseOver后发生、mouseMove最后发生；

# 鼠标事件实现

再次总结一下鼠标事件的实现吧。

侦听Dom元素的MouseOver、MouseOut事件，在这个过程中，MouseOut应该先发生，接着才是MouseOver。

在WPF中，顺序是先设置IsMouseOver，然后才是MouseEnter、MouseLeave事件的发出。

MouseEnter、MouseLeave都是Direct策略的事件——直接投递，不是Tuunel、Bubble策略的；

可能按照我先在的实现更好一些，WPF的isMouseOver、MouseEnter、MouseLeave的顺序可能是无奈之举吧。

昨天晚上想到的一个问题，如果是采用dom元素的矩形区域来判断鼠标是否位于其中的话，会有纰漏，因为在float下，容器的尺寸可能是Empty，此时进行矩形判断的话，就会出现问题了。

即使不是Empty，当孩子是float或者absolute时，父容器的尺寸也不足以判断发生在孩子dom上的鼠标位置。

# Float和Absolute

在这个情形下，可能有问题了，dom的尺寸不足以来判断孩子dom的鼠标位置了。

采用mousemove事件来判断矩形与位置的话，也就行不通了。

还是采用不够完美的解决方案吧，就是采用mouseOver，只是采用这一个事件。来解决isMouseMove、MouseEnter、mouseLeave吧。

思路是：

判断当前实现的source是否是MouseDevice中记录的mouseOver元素，如果是同一个，就什么也不做，如果不是同一个，就需要针对上一次的mouseOver元素，取消设置，同时发出mouseLeave、针对新元素设置isMouseOver、isDirectlyMouseOver，并发出MouseEnter事件。

这样唯一的不完美是，如果鼠标移出了浏览器窗口时，元素可能没有发生mouseOver事件，尤其是page上，此时是body在鼠标移出时可能就不会产生mouseLeave事件了，呵呵，bug。

~~一个做法是，创建一个div，让这个div总是比body小一些，此时如果发生了鼠标从这个div移出到body时，就会发生body的mouseOver，此时至少在WPF的范围内保证了mouseLeave的正确性。~~

# Tooltip

回顾一下

与ToolTip有关的，一个是frameworkElement中的ToolTip属性，这是一个依赖属性，类型是Object；

ToolTip对象，一个显示出来的tooltip就是这个对象的一个实例；

还有一个工具类。在ToopTipService中，定义了与ToolTip相关的一些附加属性：Placement、PlacementTarget、HasDropShadow、IsOpenProperty、VerticalOffset、HorizanalOffset、InitialShowDelay、ShowDuration；

同时定义了一些读取这些属性的静态函数。

在ToolTipService中一个关键是FindToolTipEvent，这个事件的作用是为了查找拥有ToolTip属性设置的对象。这个事件如何运行呢？首先这个时间采取冒泡的策略，还需要定义类级别上的事件侦听机制，在这个侦听中，检测当前对象是否定义了ToolTip属性，如果定义了，就返回，设置事件的Handled为true，这样设置后，就可以停止了后面的侦听，带有ToolTip属性的对象就回到了需要查找的地方，当然是在需要查找的地方进行RaiseEvent（FindToolTipEvent）了。

找到这个ToolTip属性的值后，判断是否是ToolTip类型，如果是这个类型的话。。。

当然上面的逻辑发生在PopupControlService中，这个类是以单例的形态存在吗？是的，这个单例被定义到了FrameworkElement中，在这个类中，定义了这个单例。

在PopupControlService中，处理ToolTip的步骤如下：

在构造PopupControlService的实例中，注册了一个侦听器，专门侦听鼠标事件（更准确的说是Input事件），通过MouseDevice的MouseOver记录，以及当前事件检测如果当前的MouseOver不为空，就在mouseMove中进行ToolTip处理。

找到最近的ToolTip元素，创建定时器，设置InitialShowDelay为定时间隔，此时这个地方合适的定时器应该是dom的SetTImeOut，这个时间就是InitialShowDelay属性的值，当然有默认值（400毫秒）。这个能够取消吗？clearTimeOut()也可以取消已经定义好的SetTimeOut了。

在定时到达后，执行显示Popup，然后启动ShowDuration定时器，也可以是TimeOut，如果中间有事件发生，需要快速关闭tooltip，就取消这个定时器，否则就等到定时时间到后，关闭ToolTip即可。

在这个地方，需要判断鼠标的（Up、Down、以及Click、doubleClick等事件中关闭ToolTip），显示出来ToolTip后，就在PopupControlService记录ToolTip的Control，方便取消。

显示就是创建Popup，然后把ToolTip的属性binding到这个Popup上。

到了Popup中后，此时问题就变得简单了，创建div，显示出来即可，当然需要position：absolute的设置。

在这个地方有一个关键，就是PlacementTarget的处理。

对于ToolTip来说，ToolTip所在的对象就是这个PlacementTarget，原来看懂了，才过了两天，居然又忘了，真是好脑子不如烂笔头啊！

# 1月12号

ISupportInitialize这个接口提供了两个方法，一个是BeginInit、EndInit，在BeginInit中设置了InitPending标志，此时如果添加孩子，不会触发UpdateStyle的调用，到了EndInit中，才会调用UpdateStyle的了。

# Popup

当PlacementTarget发生变化后，会触发OnPlacementTargetChanged事件，在这个事件中，会把Popup注册到PlacementTarget的RegisterPopupField定义的匿名DP中，当然作为PlacementTarget来说，可能存在多个Popup了。

在OnPlacementTargetChanged方法中，判断如果popup.isOpen为真的话，就UpdatePlacementTargetRegistration，否则就仅仅是取消之前在PlacementTarget上的注册就ok了；对UpdatePlacementTargetRegistration的调用就到了当IsOpen属性被设置为true后进行。

调用UpdatePlacementTargetRegistration的目的是为了使用PlacementTarget上的定义的资源等信息，也就说说，Popup本身并不是VisualTree的一部分，在ToolTip的场景下，但是仍然需要使用元素树上定义的资源。

# Popup

回到昨天的PlacementTarget属性的问题，一个PlacementTarget允许有多个注册的Popup，采取CommonField的方式保存到EffectiveValue中。

在设置Popup的PlacementTarget属性时，通过OnPlacementTargetChnaged回掉，来从PlacemetTarget上更新继承实行和查找资源的了。嗯，这个属性是干什么的呢？

BetweenShowDelay：当从一个控件将鼠标移动到另一个控件的间隔小于这个值的话，就需要快速显示，不再使用InitialShowDelay了。

# Tooltip

细节之后还有细节，在触发ToolTipOpenEvent事件后，如果有侦听器将Handled设置为false，意味着ToolTip将不再显示。这个细节需要注意。

# 让思绪飞一会儿

对ToolTip、Popup部分的代码理解又加深了，这个加深自然一会对其他地方的理解有触动。但是感觉还没有到达可以熟练采用dom来模拟实现的程度。

# 接下来

KeyboardNavigation部分一直是被闪躲着的，总是绕不过去的，索性连这个货也收拾了先。

# Focus Overview

一些属性

IsTabStop：是否允许tab导航到其上；如果这个属性为false的话，tab就不会到达其上。

Default Tab Order Is Sensible：默认的Tab顺序是符合实际的，tab顺序默认是按照XAML中定义逻辑树的顺序，IsTabStop设置为false的话，会被跳过，这个属性的默认值用脚趾头也能想到，应该是true。这是一个附加属性，被定义到了KeyboardNavigation中。

TabIndex：这个属性应该不会陌生，顾名思义就是为了设置Tab的顺序的；默认值是最大整数（32位）；

*AcceptsReturnProperty：*

ShowKeyboardCues：

*TabNavigationProperty：这个属性符是*KeyboardNavigationMode类型的，默认是Continue。

* Contained： Depending on the direction of the navigation, focus returns to the first or the last item when the end or the beginning of the container is reached, but does not move past the beginning or end of the container.
* Continue： Each element receives keyboard focus, as long as it is a navigation stop. Navigation leaves the containing element when an edge is reached.
* Cycle： Depending on the direction of the navigation, the focus returns to the first or the last item when the end or the beginning of the container is reached. Focus cannot leave the container using logical navigation.
* Local： Tab Indexes are considered on local subtree only inside this container and behave like Continue after that.
* None： No keyboard navigation is allowed inside this container.
* Once： The container and all of its child elements as a whole receive focus only once. Either the first tree： child or the or the last focused element in the group receives focus

这个地方的核心类是KeyboardNavigation，这个类也是以单例的形态存在，在FrameworkElement中维护创建，与PopupControlService一起。

呵呵，忽然想到了一个问题，就是屏蔽浏览器tab键的默认实现，完全交给WPF来管理导航吧，需要在所有的DOM元素上进行注册键盘事件。

大概扫描了一下KeyboardNavigation的实现，大致的思路，注册全局事件，侦听键盘事件，在tab、shift+tab、left、right等导航键盘事件中，进行导航。

分为两类，一类是通过next、previous的方式进行，另一个是通过方向位置的p判断来进行的。

后者需要判断根据元素的坐标方位确定大致的导航方式。

最终是在设置了元素的focus，当然最终也需要落实到dom元素的focus了。此时这个地方好像有点问题，设置控件为facus后，如何落实到dom元素上呢？

**Part 1: It’s Basically Focus**

SEPTEMBER 2, 2008 BY MARK SMITH[LEAVE A COMMENT](http://www.julmar.com/blog/programming/part-1-its-basically-focus/#respond)

**Focus Types**

As you may know, in WPF there are two types of focus: **logical** focus and **keyboard** focus.

**Keyboard focus** is the easiest to grasp: the element with keyboard focus is where any keystrokes will end up. Only one element can have keyboard focus at any particular time, and it’s possible that no element currently has keyboard focus (this happens most commonly when the application itself is not the activated application).

**Logical focus** is a little murkier. Logical focus represents where keyboard focus *could* go within a group of elements. There may be multiple elements with logical focus within the application and one of them likely has keyboard focus. Having keyboard focus automtically indicates logical focus but not vice-versa.

This distinction is modeled after Win32 itself – each thread has one HWND it has identified to have focus but only one of them really has input focus at any given point in time.

In WPF, logical focus is tracked and managed by a *Focus Scope*. A focus scope is created by certain elements to keep track of which element should have focus. There can be multiple focus scopes within the application – each identifying a single element that has logical focus. Focus scopes are created and managed by the FocusManager class. It exposes two attached properties and the requisite static method wrappers to access them:

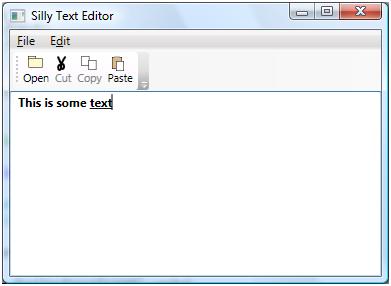
**IsFocusScope** – determines whether the current object is a focus scope.  
**FocusedElement** – returns which element child of the given focus scope object has logical focus.

It also exposes a handy method to find the focus scope for a given element – **GetFocusScope**

So what creates a focus scope and why do we need more than one? Well, the elements in WPF that create focus scopes by default are **Window**, **Menu**, **ContextMenu** and **ToolBar**. The reason we need more than one is to ensure your application works the way you expect it to. When you are typing in a **TextBox** and click a menu item, you really want focus to return to the original **TextBox** once the menu is dismissed. That’s focus scopes in action – the Window maintains *logical focus* in the **TextBox** and when you click on the menu, keyboard focus shifts to the menu. Since it maintains its own focus scope, it has its own logically focused element that WPF sets focus to (the menu and menu items in this case). Once you shift back to the window, keyboard focus moves back to that focus scope’s logical focus: the text box. Without focus scopes to track the original focus holder, WPF wouldn’t know where focus should go.

Focus scopes are also critical to command routing – often execution of commands depends upon focus. Some commands become active because a specific control which has a handler for the command has focus. Without focus scopes, we could not have menu items and toolbar buttons initiate those commands – they would steal focus away from the target, making the command unavailable. However, when WPF encounters a focus scope it checks the element that has logical focus in that scope to see if it can handle the command. If not, the command continues routing up to the parent of the focus scope.

To show all of this behavior, I have rigged up a sample application with two windows. The first is a traditional text editor window with a menu and toolbar and a RichEdit control for the content. It looks like:



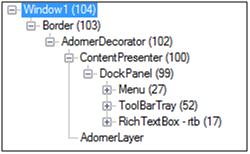
Forgive my 5-minute graphics for the buttons – I just threw them together in Visual Studio, a real project would use Blend to generate the graphics.

Regardless of it’s look, the application functions the way you expect – you can type in the text field, click buttons and select menu choices to Open, Cut, Copy and Paste content. The Cut/Copy/Paste commands are only available when the TextBox is in the appropriate state:

**Cut:** TextBox has focus and has some text selected.  
**Copy:** TextBox has focus and has some text selected.  
**Paste:** TextBox has focus and text exists on the clipboard.

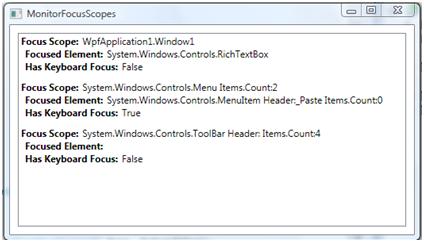
This is all an artifact of the routed command system in WPF – the TextBox has command handlers registered for**ApplicationCommands.Cut**, **ApplicationCommands.Copy** and **ApplicationCommands.Paste** and when it has focus (and the above criteria is met) those commands *could* be executed.

When you click on a button or a menu choice, it executes the command it is associated with and WPF decides which handler should be called. This normally involves walking the visual tree:

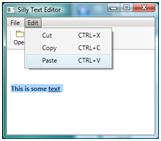


In this case, if the menu item for “Cut” were selected, it would start looking at the menu, then move up the tree consulting each parent and looking for a **CommandBinding** to handle the command. If that were the whole story then the Cut command would never happen because nothing in the visual tree above the menu has a binding to execute the Cut command. But, of course, it’s not the whole story – in this case, WPF sees that the Window is a focus scope and so it gets the logically focused element from the Window and looks for a CommandBinding there. That happens to be our RichTextBox – which is where the command ultimately gets handled.

To see all of this in action, the second part of the application is a focus scope monitor window.



It shows all the known focus scopes and what the active focused element is, as well as whether that element has keyboard focus. The window is live so as you click around in the text editor you can see focus shifting and changing. The element with keyboard focus is the Paste menu item – notice that the rich text box is the logically focused item for “Window1” but does not have keyboard focus. Once you select the menu choice (or cancel) then keyboard focus shifts back to Window1 which assigns it to the RichTextBox.



If you’d like to play with this sample, you can get the full source code [here](http://www.julmar.com/content/binary/SimpleEditor.zip).

In the next blog post, I’ll write about how you can programmatically assign and control focus in code and XAML.

**Part 2: Changing WPF focus in code**

SEPTEMBER 4, 2008 BY MARK SMITH[LEAVE A COMMENT](http://www.julmar.com/blog/programming/part-2-changing-wpf-focus-in-code/#respond)

In the [last post](http://www.julmar.com/blog/programming/part-1-its-basically-focus/), I wrote about how focus is generally managed in WPF – we have focus scopes to track a single element within that scope for logical focus, and then one of those elements is given physical, or keyboard focus.

Now, let’s talk a little about how you can influence that programatically. First, you can always determine which element has logical focus in your application through the FocusManager.GetFocusedElement method — pass it the window in question and it will return which element has logical focus in that window. Remember that logical focus != keyboard focus at all times — toolbars and menus track their own focus so if you are currently interacting with a menu then the menu has physical focus. But in general, the following code will tell you which element WPF thinks has focus in the window:

IInputElement focusedElement = FocusManager.GetFocusedElement(thisWindow);

To determine whether this element has keyboard focus, we can check the IsKeyboardFocused property – if it’s set to true, then that element currently has the keyboard focus (as well as being the logical focus for that focus scope).

Keyboard focus is most often set through runtime activity – the user clicks on an element, or uses the TAB key to move around the UI. You can also set it programatically a couple of ways. First, there is aKeyboard class in WPF which exposes several methods and properties. There is aKeyboard.FocusedElement read-only property which returns the current keyboard focused element, and there is a Keyboard.Focus method which attempts to change keyboard focus. It returns the element that now has focus – so you can check to see if your request was fulfilled or ignored. So, for example, you can change focus during your application initialization:

void OnLoaded(object sender, RoutedEventArgs e)  
{  
Keyboard.Focus(firstTextBox);  
}

Notice that we uses the Loaded event – this is because no focus requests will be accepted prior to the element being initialized and loaded. That’s the first place in the application where you can make focus changes.

When would setting focus fail? Well, it can fail for a lot of reasons, but the most common are:

The element has Focusable = false

The element has Enabled = false

The element has IsVisible = false

The element has not been loaded yet

The currently focused element will not release focus.

That final one is important, changing focus involves potentially taking it away from an existing element – they receive a PreviewLostKeyboardFocus and LostKeyboardFocus event. If they handle the preview event, focus will not change.

You can also manipulate focus through programatic keyboard navigation – simulating the user pressing TAB to cycle through the focusable elements. This is controlled through the KeyboardNavigation class which is used when the user presses a key that changes focus (TAB, SHIFT+TAB, Up, Down, etc.).

Controls can set a TabIndex property assignment which determines the tabbing order. The default is to tab through them in order of declaration. You can also use the KeyboardNavigation.TabIndex attached property which works for any element – not just controls.

To control navigation, the KeyboardNavigation class has an attached property TabNavigation allowing you to change how navigation occurs within a container. You can set it to:

Continue – each focusable element receives focus and the container is exited when the edge is reached.

Cycle – focus does not leave the container but wraps around the edges

Once – the container itself is treated as a single focusable element where only the first child receives focus

Local – uses TabIndex locally within the container – independant of any outside elements.

Contained – focus statys in the container but does not wrap (stays at edges when top/bottom are reached)

None – no keyboard navigation allowed in the container

The default is Continue, but you can set the attached property on any element to change it for that element and any children. To see this in action, paste the following into your XAML editor of choice and change the ComboBox while tabbing through the TextBlock elements.

<Window xmlns=”http://schemas.microsoft.com/winfx/2006/xaml/presentation”  
xmlns:x=”http://schemas.microsoft.com/winfx/2006/xaml”  
xmlns:sys=”clr-namespace:System;assembly=mscorlib”  
Title=”Simple Focus”>

<Window.Resources>  
<Style TargetType=”TextBox”>  
<Setter Property=”Margin” Value=”10″ />  
<Setter Property=”Width” Value=”100″ />  
</Style>  
</Window.Resources>

<StackPanel>  
<ComboBox x:Name=”tabStyles” SelectedIndex=”0″ Focusable=”False”>  
<sys:String>None</sys:String>  
<sys:String>Continue</sys:String>  
<sys:String>Cycle</sys:String>  
<sys:String>Once</sys:String>  
<sys:String>Local</sys:String>  
 <sys:String>Contained</sys:String>  
</ComboBox>

<TextBox TabIndex=”1″ />  
<TextBox TabIndex=”2″ />  
<StackPanelKeyboardNavigation.TabNavigation=”{Binding ElementName=tabStyles,Path=SelectedItem}”>  
<TextBox TabIndex=”1″ />  
<TextBox TabIndex=”2″ />  
</StackPanel>  
</StackPanel>  
</Window>

You can have the system ignore specific elements (but still allow them to have focus) by setting theKeyboardNavigation.IsTabStop=”false” attached property. This will cause keyboard navigation to “jump” over the control as if it were not present.

Three methods are exposed by UIElement and FrameworkElement to programatically shift focus: Focus,MoveFocus and PredictFocus.

To force focus to a specific element, you can call Focus on it. For example, above we set the keyboard focus by calling Keyboard.Focus(), but the same effect can be achieved like this:

void OnLoaded(object sender, RoutedEventArgs e)  
{  
firstTextBox.Focus();  
}

This method attempts to set focus using Keyboard.Focus(). If that fails, but the element is Focusable and enabled, it finds the focus scope for the element and sets logical focus there (so that keyboard focus will eventually end up on the control).

FrameworkElement.MoveFocus is used to change the keyboard focus in the application using the same algorithm as the TAB traversal. You pass in the direction (specified through a TraversalRequest object) and the method returns true/false to indicate success. Under the covers it actually uses the KeyboardNavigation class, but it’s an easy way to push focus around the window:

firstTextBox.MoveFocus(new TraversalRequest(FocusNavigationDirection.Next));

PredictFocus works the same way, but instead of actually shifting focus, it returns what would be the focused item if you were to execute MoveFocus.

So, up to this point, we’ve seen a lot of code to change focus. However, the most common request is to set initial focus to a specific control – remember that WPF doesn’t do that by default. You can do it in code, just like the above example where we use the Loaded event. Or, it turns out you can do it in XAML too. The key to remember is that the FocusedElement of the main focus scope (the Window) is the one that will get initial focus. That is (by default) null, but you can set it in XAML using the attached property syntax. Using the above XAML example, we can supply a name for one of the TextBox controls and then a little data binding magic to set that onto the Window:

<Window xmlns=”http://schemas.microsoft.com/winfx/2006/xaml/presentation”  
xmlns:x=”http://schemas.microsoft.com/winfx/2006/xaml”  
xmlns:sys=”clr-namespace:System;assembly=mscorlib”  
Title=”Simple Focus”  
FocusManager.FocusedElement=”{Binding ElementName=tb2}”> …  
<TextBox TabIndex=”1″ />  
<TextBox x:Name=”tb2″ TabIndex=”2″ />  
…  
</Window>

Now when you run the application, focus is placed into the second text box in the window. This technique works great as long as the element you want to assign focus to is declared here in the same XAML file. However, a popular way to develop WPF applications is to separate out chunks of UI into separate UserControls. When you do that, the above trick fails — even if you put the FocusManager.FocusedElement binding into the UserControl!

How we solve that is what we’ll look at in the next post! Stay tuned…

# 1月13号

继续Focus部分的代码阅读。

KeyboardNavigation是处理键盘焦点相关的代码，FocusManager是处理逻辑焦点的；

一个场景：

对于Button来说，如果其内容包含了UIElement的话，如果设置其拥有焦点，就需要好好的斟酌了，假设有一个TextBox的孩子，Button拥有焦点，应该是这个TextBox拥有焦点才对。而KeyboardNavigation也好像是这么处理的。

在DOM上实现WPF的话，当弹出一个div作为popup后，此时一个有趣的现象是，tab可以导航在popup上，然后又可以回到popup后面的页面上（或者父Popup上）；

WPF通过在一个Window上构筑自管理的一系列控件，来代替Window原生的控件，自然需要接管Window上的原生消息，经过处理后设置到响应的控件上的，在这个地方，需要记录HitTest的结果，以及键盘焦点，以便进行消息投递。把消息投递到这个控件上。

因此，KeyboardNavigation中Focus的本质，就是进行这个记录而已；

到了DOM中，其核心原理，应该是，通过setFocus来设置拥有焦点的dom元素，此后的键盘事件就发送到这个dom元素上了。在操作dom元素的焦点过程中，处理焦点的转移事件（PreviewGetFocus、GetFocus、PreviewLostFocus、Focus四个事件）。同时还需要设置isFocus等属性。这个地方的顺序，可以按照WPF的处理顺序。到了这个地方，可以说，dom事件仅仅被当成了输入，被完全转换为WPF的逻辑事件了，而用户应该就是面向这些逻辑事件进行编程的。

焦点的设置是在KeyboardDevice中完成的，分别有Focus、TryChangeFocus、ChangeFocus等几个方法。

在UIElement中，有一个Focus方法，这个方法设置逻辑焦点，如果当前键盘焦点元素在一个FocusScope中，同时也改变键盘焦点到这个元素上。

TabNavigationProperty:

Controls the behavior of logical navigation on the children of the element this property is set on.

控制当前元素孩子们的逻辑导航方式。

ControlTabNavigationProperty:呵呵，通过看源代码，我明白这个属性的含义了，当Tab、Shift+Tab过程中，如果按下了Ctrl键，那么就是这个属性定义的导航方式了。

这个是微软MSDN的权威解释：

Gets or sets the logical control tab navigation behavior for the children of the element that this property is set on.

*DirectionalNavigationProperty：明白了COntrolTabNavigationProperty后，这个属性自然也就不在话下了，就是当通过Left、Up等箭头键导航时，遵照的导航方式。*

这个是微软MSDN的权威解释：

Gets or sets the directional navigation behavior for the children of the element that this property is set on.

*嗯，微软的设计确实人性化，但是未免也太繁杂了，这个特性居然两自己这个资深业内人士也不知道。*

*刚才搞错了，箭头导航键在ListBox中时，不是很常见吗，当然知道了，只是在这个层面忽然短路了而已。*

*真是不错，一下子明白了三个属性的含义；*

*这三个属性都是针对容器内的孩子们来说的；*

再次回到这个属性对应的枚举来理解一下：

KeyboardNavigationMode：

Contained：在容器内按照一个方向导航到达头部后，就不再移动，只能向相反的方向移动；

Cycle：在容器内，到达头部或者尾部后，进行循环；

Continue：遍历每一个IsTabStop设置的元素，到达边缘后，就跑到容器的下一个元素；

只要不是Continue的模式，就是Group组态；

None：不允许在此容器内进行键盘导航；

Once：只有容器内的一个元素拥有焦点，然后就需要跑出到容器外；

# KeyboardNavigation

关于ShowFocusVisual（）方法，在这个类中，当拥有焦点后，就创建FocusVisualAdorner来装饰当前的元素；

FocusStyle就是应用到这个元素上的。

从这个意义来看，既然Dom的HitTest不能自己处理，意味着无法通过layer的形式来实现FocusStyle的特性了，因此这个特性有关的代码是需要删除的。

# 鼠标和Focus

当鼠标点击后，焦点是如何处理呢？

分为两类：

一类是ButtonBase为代表的，当发生鼠标点击后，会让ButtonBase拥有焦点；

另一类是对鼠标点击不需要涉及焦点操作的；

TextBox呢？

确实是如此，当TextBox发生鼠标点击后，到底是先发生focus呢？还是先发生mouseDown呢？应该是MouseDown先发生吧。在这个地方应该接管dom的focus事件，一遍能够拥有焦点的问题。

测试一下input（type= text）的mouseDown和focus事件；看看能否阻止mouseDown的默认行为呢？

测试发现，在Input中注册mouseDown事件侦听，在处理函数中，如果组织默认行为，呵呵，居然就不能拥有焦点了。但是为什么click还会发生呢？

# Focus

在DOM中实现DOM的focus的话，需要重载UIElement的focus方法，在这个方法中，根据不同的控件类型，分别设置dom的focus方法。

原来设想的每一个dom元素都可以接管键盘事件的想法有点问题，在WPF中，panel就是不能接管键盘事件的了。

# Button

当发生click后，调用了UIElement的focus方法，在这个方法中，调用Keyboard的focus方法。

# 几个概念

Logical Focus：一个FocusScope中的拥有焦点的元素，如果当前FocusScope拥有键盘焦点，那么逻辑焦点和键盘焦点就是一个元素。

Keyboard Focus：键盘 事件投送的元素。

MouseOver、IsKeyboardFocus：一个是鼠标当前位于其上的元素，另一个是当前拥有键盘焦点的元素。

键盘导航：通过tab、shift+tab、left、right、up、down移动键盘焦点，称之为键盘导航；

FocusScope、NavigationGroup（Container）；

TabNavigationProperty

ControlNavigationProperty

DirectionNavigationProperty：箭头键导航的模式，假设ListBox中，当一个ListBox接收到键盘事件时，表明这个ListBox已经拥有了键盘焦点，此时如果发生了箭头键的键盘事件，意味着就需要进行当前元素的子元素进行键盘导航了。

TabIndex

IsTabStop

Focusable

Enable

Visible

# ListBox

关于键盘导航的一些设置：

IsTabStop为false；

TabNavigationProperty：Once；

DirectionNavigationProperty：Contained；

# Ctrl+Tab

这个快捷键的作用是在MDI中进行文档切换；这个快捷键在浏览器上没有什么用武之地；

# Html Focus

按照Html规范，一个dom元素要想能够拥有焦点，tabIndex被设置，或者disable没有被设置；

通过setFocus就让dom元素拥有了焦点。在grid布局中，这个focus问题需要被注意。

在一个dom的焦点事件中设置另一个元素的焦点，可能会有问题。

Document.activeElement来获取当前文档中拥有焦点的元素。

在HtmlInputElement元素上有Disable属性，这个属性并不是HtmlElement的；

# 1月14号

看看TextBox中的UpdateSourceTrigger处理机制；

# 一个奇怪的语法：

今天下午调试一个js函数，奇怪的是总是返回undefined

Function f(){

Var r = 1;

Return

r；

}

这个奇怪的bug，叫做插入分号陷阱。

# 键盘导航

经过几天的奋斗，键盘导航终于出来效果了，但是还需要继续深入的测试才行。

TextBox的导航问题；

# 设置初始焦点

如果在Arrange前设置焦点的话，此时虽然设置到了dom上，但是由于dom没有被添加到文档树上，因此这个焦点设置到Arrange后，就丢失了。

可能还需要根据Keyboard中的焦点元素再设置一次。

# ButtonBase

当拥有焦点时，此时SpaceKey可以产生Click事件了。

# 1月15号

继续研究键盘导航问题。

设置初始焦点问题，如果还没有Arrange的话，此时的Visible就是false，是从VisibleCache中获取的。

# DirectionNavigation

从WPF的实现来看，采用了位置判断的方式，来查找接下来的位置，但是这个策略在dom中可能会失效，因为一个dom当其孩子是Absolute或者float时，其尺寸为零；

ListBox也是这个策略吗？

# ScrollView

看样子这个鬼玩意也是绕不过去了。

虚拟视图部分可以先不去管它；

# Html的滚动

呵呵，scrollLeft、ScrollTop是容器上的属性，通过设置scrollTop和ScrollLeft来实现内容在容器中的可见；

子元素的offsetTop和ScrollTop之间是什么关系呢？可以读取孩子的offsetTop，然后设置父容器的scrollTop，从而实现与孩子之间的互动。

ScrollHeight：可滚动内容的大小；ScrollWidth：可滚动内容的宽度；

OffsetWidth：视口的宽度；offsetHeight：视口的高度；

滚动的本质就是设置视口的位置，从而变化内容的可见区域，达到滚动的效果。

在OnScroll事件中，Event本身没有提供什么有价值的信息，也就说，如何设置滚动的单位；

通过获取孩子的offsetTop，然后设置scrollTop为这个offsetTop，就可以让孩子处于可见区域内。也就是实现WPF的MakeVisible效果。

Div的滚动事件基本上不需要自己参与吧；

滚动条滚动内容的特性：鼠标点击、滚动滚轮都已经实现了内容滚动。

考虑到ListBox的话，还缺少的是当箭头键在子元素上的导航。当然这个导航会需要将不可见的内容变得可见；而现在的滚动条，对于箭头键的默认行为是滚动内容区域，并没有处理子元素的键盘导航。

# 控件的实际尺寸

既然dom元素上有与实际尺寸有关的信息，实现控件的实际尺寸理论上来说是可能的。

通过实际尺寸的使用，理论上应该可以实现按照位置来查找的导航了；

# Dom的Style设置

从控件的属性设置对应dom的style属性。

在FrameworkElement中的几个属性：

Width

MinWidth

MaxWidth

Height

MinHeight

ManHeight

其他两个在dom实现中没有意义，因此可以只需要width和height两个属性，对应于style的width和height；并且需要响应这两个属性的变化；

嗯，这个地方说明对css还是不够掌握，在css中也确实存在minwidth之类的属性。

# Cursor

Cursor属性：可以对应dom吗？

Object.style.cursor = xxx;的形式来设置光标形状；在DOM中光标是一些预设的值；

这个属性可以了；

# Margin

在WPF的实现中，Margin采用是Double类型，这个类型有点问题，到了css中，因为还要携带单位，因此这个地方有点问题。当然可以认为这个是pt的值；也就是只有一个逻辑单位；可以按照WPF的规范，采用WPF的单位了，这样的话，可以认为小于1就是百分比，大于1就是逻辑单位了。

这个96分之一，就是最小单位；

嗯，如果牺牲掉百分比这样的特性的话，可能会遭到诟病；

# TextBox

对于键盘事件尽可能少的影响默认行为，对于TextBox来说，如果对于任何键都取消了默认行为的话，这个TextBox两backspace都不行了。

如果使用了TextBox的默认行为，意味着，就不能有双重行为了，例如：箭头键的键盘导航。

为了这个特性，在添加事件中是否阻止事件的默认行为，最好有控件本身来决定。

# 获取控件矩形

在ListBox中，箭头导航子元素时又碰到了获取控件矩形的问题。嗯，解决它；

# 1月16号

ListBox的键盘焦点还在继续；

把Rect、Size、Point重新检查处理了一遍；

# 小成果

键盘导航居然跑通了，在CheckBoxTest。Html上，箭头键导航居然跑通了。呵呵，真是功夫不负有心人。

# TextBox

默认行为问题需要解决了。

在EventListenerManager中，可以把添加事件的方法交给控件自身；在FrameworkElement中，实现基本的，到了TextBox中重载这个方法，从而实现去掉tab的默认行为，但是保留其他的默认行为。

# DOM事件侦听

原来的方案，通过一个工具类来处理的方式，不是很灵活，需要重构；

由于整个基类分为两个部分，因此，分为UIELement和ContentElement两个地方，在这两个地方设置基类，AddDomEventListeners（）的方法。

在子类中进行重载，在FrameworkElement中重载可以覆盖Shapes，Image等

特殊的重载在TextBox中了；

把每一个事件的注册分开，这样应该会更加灵活。

需要照顾一下基础服务部分：Popup等、MouseOver等、焦点、快捷键等的处理；增加一个PreProcess和PostProcess两个拦截器，来模拟WPF的事件机制。

# WPF的事件机制

HwndMouseInputProvider：将win32的底层鼠标消息，转变成WPF事件；

HwndKeyboardInputProvider：将win32的原生键盘消息转变成WPF事件；

HwndWrapper:

Dispatcher:在这个类中，找到了熟悉的一些代码片段。循环获取消息，然后分发消息。对传统的windows编程的封装。

对于原生事件：经过三个步骤来处理

PrePrecess：预处理

Process:发出路由事件

PostProcess：后处理

# 一个副作用

当鼠标离开浏览器窗口时，此时mouseover可能会有点问题，那就通过window的事件来处理吧，如果window失去了焦点，可以附带处理一下mouseover的断裂了。

# Window上的事件

Onfocus：当窗口获得焦点时，发生，onblur当窗口失去焦点时发生。嗯。这个东东可能会有大用处。

嗯，body元素上没有这两个 事件，看样子在page的处理上，需要采取特别措施，不能简单的让body拥有焦点即可。

# 一个网站

Jsil.org居然把il直接翻译为可以运行的js，然后在浏览器上运行。呵呵，看样子，对WPF感兴趣的人真不少。

# Javascript

最大整数：0x40 00 00 00 00 00 00

# 1月17号

继续实现Calendar控件；

现在发现grid这个控件实现的不正确，先花些力气完成这个。

# FrameworkElementFactory

在这个里面无法设置Resource的属性。嗯，这个是一个讨厌的问题；

# Style设置

如果属性是来自于继承的话，在style上这个属性就无需设置了，充分利用css的继承特性就好了。如何进行判断呢？此时获取的source应该是inherited吧。

# Grid

可以先把Tr完成出来，因为已经可以看出有多少个了；

创建一个二维数组，先把定义按照顺序填入这个二维数组，每一个单元格保持一个集合，用来记录放置在这个单元格中的孩子。

单元格已经具备了span属性，接下来可以直接使用了。

在这个单元格中，已经被span的单元格，其内容设置上标志，来区分内容为空还是span的单元格。

接下来，循环这个单元格，以此创建tr、和td、对于cell为span的单元格，就不再创建td了，而对于内容为空的单元格，则需要创建特殊的内容，来填入这个单元格。

这个地方不考虑支持WPF的比例方式。

对于没有定义行列的grid，还是采取div'的方式，在这个div中进行显示子元素。

# SVG

Viewbox的意思，设置视口的坐标。

位于其内的形状，就是采用这个坐标系的。

例如：

<svg width = ‘100’ height =100 viewBox=’0 0, 1000, 100’>

<rect width=900, height=900>

</svg>

此时保证rect能狗绘制到svg元素中。

这个属性能够在翻译WPF的shapes的过程中使用吗?

可以把Stretch这个属性修改为ViewBox属性，从而可以利用SVG的ViewBox属性了。

# ViewBox

还是引入一个这个的控件，只是这个控件并不对应dom元素，而仅仅是设置孩子元素的高宽比例而已。

# Calendar

呵呵，终于出来可dayButton部分，但是上面的星期没有显示出来；

# SVG

今天算是碰上硬茬子了。SVG没有zindex属性，意味着如果与其他控件重叠到一起的话，就会把其它控件遮挡住，此时就没有机会在于被挡住的元素进行交互了。

如果要实现交互的话，就需要把svg当作背景了，在编译时，把svg的图形创建变成一个svg文件，然后制作背景给另一个元素，从而实现堆叠。如果这样的话，意味着这个svg就不能响应属性变化了，尤其是在属性进行了binding时。

在实现image的背景时，就可以实现采用svg的背景了。

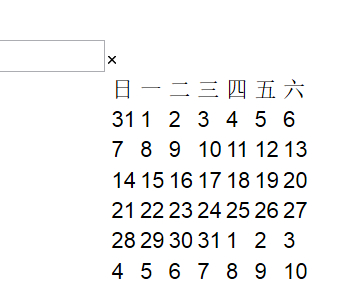
看样子也只能实现采用透明方式的堆叠了，呵呵，有点不是很爽啊。

如果非要进行交互的话，例如：可以直接在shape上进行事件侦听，实现交互，倒也不失为一个解决方案了。

目前可以先不采用矩形在calendar中使用Rectangle了。

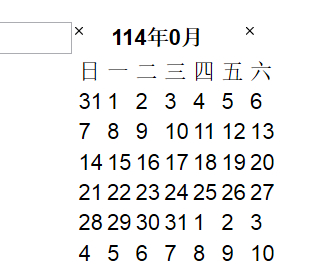
# 1月18

呵呵，日历居然算出来了。

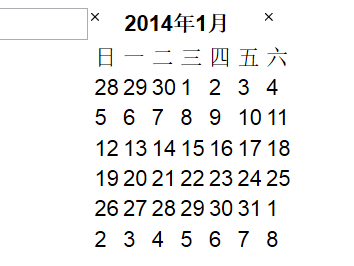


上面是日历的一个最新摸样；

把上面的三个按钮也整出来。

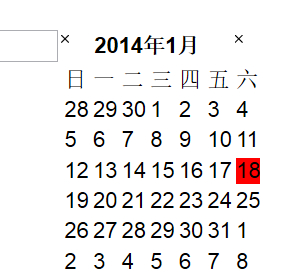


三个按钮整出来后的样子；



设置当前日期后的样子；

选择选择当前日期：比较当天的计算，取出当前时间，与日历中的每一个day进行比较，这个地方，应该只是比较日期部分就好了。算上时间就不严谨了。



这个是设置当前日期后的样子。

# 接下来

处理鼠标事件的侦听代码；

前进后退按钮也能正常进行了；

# 一个问题

忽然想到一个问题，那就是svg是否可以会被绝对定位的div遮挡住呢？

# SVG和Canvas

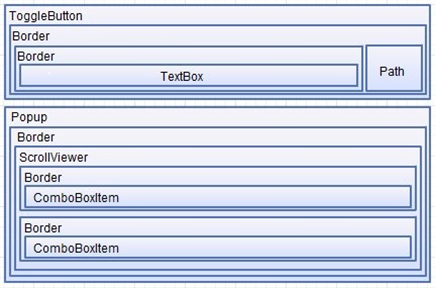
Svg是dom元素，支持对齐文字的选择吗，也就说，如果svg内有文字的话，是可以通过鼠标进行选择的；

果然，canvas的text是不可选择的，但是canvas是可以被div遮挡的。当然canvas也是可以响应鼠标的，如果仅仅是从模拟WPF的角度看，可能canvas会更加合适吧。当然初始化脚本紧随其后；但是问题是刷新会出现问题了，尤其是与DP的binding后；也没有关系，因为来来回回就是几个图形而已，而不是Canvas的巨大API，这样的话，还是可以得，只是在属性变化后，重新绘制而已。

# 1月19

接下来攻克ComboBox吧；

其模板的组成结构：



是两列：第一列显示ContentPresenter、TextBox，用于显示当前的文本或者SelectedItem；第二列显示下下拉箭头；因此说一个ComboBox有很多个元素组成。呵呵，有点复杂，很担心这种复杂性，可能导致系统可用性降低；

DropdownButton

ContentPresenter

TextBox

Popup

Popup的Placement属性默认是Bottom——控件底部，自然也就正好满足ComboBox的要求。

Popup的IsOpen绑定到了ToggleButton——ComboBox的IsDropDownOpen属性上，通过TemplateBinding方式绑定；

ComboBox有一个属性MaxDropdownHeight属性，用来设置Popup的高度的；

在其中，ToggleButton中，其isChecked属性与ComboBox的IsDropDownOpen属性banding到了一起，呵呵，这个ComboBox的运行机制基本上算是掌握了；

# ItemsPresenter`

这个类的意思是项集合表现器，在没有进行Grooup的情况下，这个类自动从父元素（TemplatedParent）上获取ItemsPanelTemplate，作为自己的Template，展开这个Template后，核心就是创建一个Panel，这个Panel就是集合的项容器了。

在ComboBox中

<Style TargetType="{x:Type ComboBox}">

<Setter Property="Control.Template">

<Setter.Value>

<ControlTemplate TargetType="ComboBox">

<Grid>

<ToggleButton Name="ToggleButton"

Template="{StaticResource CBCustomToggleButton}" />

<ContentPresenter Name="ContentSite"

Content="{TemplateBinding ComboBox.SelectionBoxItem}"

ContentTemplate="{TemplateBinding ComboBox.SelectionBoxItemTemplate}"/>

<TextBox x:Name="PART\_EditableTextBox"

Foreground="{TemplateBinding Foreground}"

IsReadOnly="{TemplateBinding IsReadOnly}"

Visibility="Hidden" />

<Popup Name="PART\_Popup"

IsOpen="{TemplateBinding ComboBox.IsDropDownOpen}">

<Grid Name="DropDown"

MinWidth="{TemplateBinding FrameworkElement.ActualWidth}"

MaxHeight="{TemplateBinding ComboBox.MaxDropDownHeight}"

SnapsToDevicePixels="True">

<Border Name="DropDownBorder"

Background="{TemplateBinding ComboBox.Background}">

<ScrollViewer Margin="4,6,4,6">

<ItemsPresenter KeyboardNavigation.DirectionalNavigation="Contained" />

</ScrollViewer>

</Border>

</Grid>

</Popup>

</Grid>

<ControlTemplate.Triggers>

..

<Trigger SourceName="PART\_Popup" Property="Window.AllowsTransparency" Value="True">

<Setter TargetName="DropDownBorder" Property="FrameworkElement.Margin" Value="0,2,0,0" />

</Trigger>

<Trigger Property="ComboBox.IsEditable" Value="True">

<Setter Property="KeyboardNavigation.IsTabStop" Value="False" />

<Setter TargetName="PART\_EditableTextBox" Property="UIElement.Visibility" Value="Visible" />

<Setter TargetName="ContentSite" Property="UIElement.Visibility" Value="Hidden" />

</Trigger>

</ControlTemplate.Triggers>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

上面就是ComboBox的一个模板，在这个模板中ItemsPresenter虽然是通过Popup嵌入到了ComboBox中，但是其TemplatedParent还是ComboBox，因此，自然也就可以获取ItemsPanelTemplate了。

获取到这个模板后，在ItemsPresenter展开模板后，即可拥有了Panel，在进行Measure时，获取Children时，此时通过Generator创建了孩子，然后再展开这些孩子。于是控件就进入了运行状态。

由于Generator是从ItemsControl获取的，自然也就没有问题，创建孩子的过程中，这个ItemsPresenter仅仅是一个二传手的作用；

# ComboBox

经过一天的调试，ComboBox总算是具备了基本的样子。

点击下来按钮后，出现了Popup；居然三个浏览器能够有一致的特性，真是不容易的一件事情。

总结：

ToggleButton的IsCheck绑定到了ComboBox的IsDropDownOpen，然后Popup的IsOpen通过TemplateBinding到了ComboBox的IsDropDowmOpen属性上，这样当点击ToggleButton后，触发Toggle，开关IsCheck、进而改变ComboBox的IsDropDownOpen，进而设置Popup的IsOpen，然后再OnIsOpenChanged事件中，显示这个Popup；

这个地方有点问题：

就是这个PlacementTarget的属性，在Tooltip情况下，这个属性就是ToolTip来源的Owner，对于在ComboBox的场景下，此时Popup已经父亲了（其TemplatedParent是ComboBox）；

这个地方还需要进一步体会、领会才好；

在ToolTip的情形下，Popup是在什么时候附加到Dom的呢？应该是在显示时，动态附加到PlacementTarget的dom上的。然后进行绝对定位；

在ComboBox的情况下，可以把Popup的Dom添加到对应的逻辑视觉父亲上，但是不显示，只有到了需要显示时才显示这个Popup的div了。

# 需要测试列表的当前选择项

# 1月20

任务：需要研究一个Loaded是怎么一回事，呵呵，又到了更深的水域了。

按照Loaded的语义：在完成布局、Render后，可以进行用户交互的时候，就是发出这个Loaded事件的最佳时机。

嗯，这个事件还是有用的，例如：可以在这个事件中设置初始焦点；

# Arrange

在设置一个UIElement到HwndSource时，此时会调用RootVisual的Arrange方法，在这个方法中完成了布局，递归调用孩子们的Measure和Arrange，当这个方法返回时，意味着VisualTree已经创建好了。

此时其实就是发出Loaded的最佳时机；

在添加一个UIElement到HwndSource时，就会触发Loaded。从代码来看，触发Loaded事件是在Arrange和Measure之前，也就说，此时模板还没有展开。

# Loaded

其实就是解析完成，并且把所有元素都连接好了——当然是针对XAML已经变成运行时了，此时，抓住根，挂接到Win32

的Window上，此时就会触发Loaded事件，这个Loaded事件，自然是自顶向下的进行了；

因此在自己实现时，也要符合这个约定，解析好后，变成js，在浏览器上执行这个脚本，在Arrange之前，就可以出发这个事件了，这个地方还是涉及到此时对dom操作的有效性，尤其是交互有关的，如果在js运行创建元素树的过程中，同步创建dom的话，可能就没有此类的问题，刚才说的问题是调用setFocus之类的dom方法时，由于dom还没有连接到documents中，此时这个调用会丧失了效果。

# Page的导航

每一个页面既然存在名称空间，在编译为html时，也是遵照这个名称空间的约定，自动对应为目录结构，从而保证当用户输入一个url的某个模式时，就会自动使用这个开始page了；

在选择导航时，也就可以统一为标准的url了。这样可以方便与其他系统的集成了；

其实就是一个普通的html，只是这个html的内容是有其中的js动态构造的，当然这个js是可以单独在一个地方，也就是可以独立为一个与html同名的js文件，在html中倒入这个js而已；

此时一个问题就变成比较重要了，就是两个页面之间的数据转换。当然，页面导航也是可以通过post来进行的，此时意味着存在页面提交的数据了。如果要导航的页面是一个html文件，此时自然没有servlet什么事，因为没有经过servlet。

WPF是如何处理的呢？

三个手段

通过代码Navigate（Page、Parameters）；

通过Page的LoadedCompleted事件；

通过Application来进行；

在js中，当然，也可以通过new Page的方式来直接进行，只是此时不能是XAML，而是Page对应的类名称。

1.首先new一个page对象：

Page2 page2 = new Page2();

2.通过NavigationService.Navigate传递参数

this.NavigationService.Navigate(page2,"frompage1");

在Navigate方法中，来处理这个参数，由于此时都是基于代码的，应该是没有问题的，关键是Navigate方法的处理，在这个里面，需要删除body'的当前内容，然后由新的page来重新填写内容；如果要交接数据就可以增加初始化、或者loaded事件侦听，在这个里面接收传递进来的数据了。

如果是通过form提交的方式来进行呢？

此时应该就需要通过Servlet，在这个Servlet中，返回需要的页面。

在jee的标准中，是通过提交的数据来构造新的页面的，但是到了summer中，就不行了，当然可以在servlet中动态构造js，来使用提交的数据，呵呵，也是一个办法。

当然，也可以通过DataProvider来从服务器上获取数据，在获取的过程中使用Session数据来完成数据传递；

可以把这个数据序列化到html中

# XAML解析

在变成js的过程中，需要BeginInit、EndInit的调用，此时就保证Initialized事件的正确性了；因为这个事件是在添加到逻辑树上时发生的，如果按照InitializeSupport接口的约定来调用的话，就可以较好的保证初始化事件了；

需要把SourceResource变成直接的调用，也就说在创建元素树时调用，一旦到了元素树种，此时已经没有StatisSourceExtension之类的物件了。

WPF是在Application中设置MainWindow，这个属性在基于浏览器的实现中不再合适，需要改为startPage；

# 远程调用

类似于GWT的远程调用。

在XAML的代码中，调用一个接口，这个接口有标注，于是在编译为js时，编译为ajax的调用；

# DatePicker

呵呵，又是一天的奋斗，不过这个东东总算是出来大概的模样了。

发现的问题：

1. 动态资源binding需要进一步测试，在这个控件上还行不通；
2. 一个严重的问题就是静态属性问题，在面向对象的语言上，定义在基类的静态属性，只要不是私有，子类就可以看到，但是在js的类实现中，却不是这样的，基类中的非私有静态属性，在子类中是不可见的。

可以通过把这些非私有的静态属性再次设置到子类上来实现面向对象的特征。呵呵，无谓的编写一些代码；

# 代码太庞大了

比较起来jquery的实现，现在的代码太庞大了，确实需要狠狠的进行减肥才行。

# Grid

是否下决心把Grid也实现呢？

下决心实现出来看看，至少可以对现在的代码进一步进行检查吧。

这个部分的代码至少有几万行，呵呵，还真是一个挑战呢？

不过从其特性来看，还是很有吸引力的。

# Toolbar

是否实现这个toolbar呢？

# 1月21

接下来还是先梳理一下目前已经迁移的，进行查漏补缺，同时进一步加深理解，过了这一遍后，应该可以进行简化了，然后就应该是基本上固话的UI了。

前一段时间一直在拼命向前跑，很多细节都被忽视了。

几个要解决的问题：

1. ItemsControl部分的选择、动态添加、删除；
2. Binding部分的Validation、MultiBinding、BindingGroup等；
3. Shape部分的pen和Brush的属性
4. 背景图片、渐变；
5. Margin、Border
6. 圆角
7. UpdateSourceTrigger的LoatFocus和PropertyChange

# 今日任务

1. Margin：这个地方涉及到的一个烦恼是css的单位是丰富的，而到了wpf中，此时单位就比较单一了，就是WPF单位（96分之一英寸）。当我们定义了大于1的数字的话，其实就是像素，只是我们在进行逻辑单位转换时，是按照每英寸96点的方式来计算而已。

可以约定，对于0-1之间的数字，认为是百分比，之外的数字都是非百分比。统一采用pt作为逻辑单位，采用px作为逻辑单位，但是要照顾到不同的设备。

采用wpf的单位，在输出css时，需要转换成像素来使用。应该可以在页面上设置是采用逻辑单位还是物理单位，这样在输出css时，就可以遵照这个选项来进行了。如果是采用逻辑单位，就采用pt，如果是采用物理单位就采用px；

现在可以先实现统一采用px和百分比的；在margin中，对于-1~0之间的数字，采用其绝对值；

Margin、Padding、Border、CornerRadius、width、height、max（Min）Height、max（Min）Width等都采用这个策略来处理。

1. 在XAML中，统一采用WPF单位和百分比。只有这两个尺寸单位了。
2. 背景图片

# 边框

在WPF中，边框是通过Border对象来进行的，到了CSS中，由于这个Border是BoxModel的一部分，因此，这个Border控件对象实在是没有必要，但是为了能够binding，当然需要一个Border对象；

在WPF中具有如下的属性：

Background

CSS3的Border

Border-image属性

* border-image-source：指定image的source
* border-image-slice：
* border-image-width：
* border-image-outset：
* border-image-repeat：

在FrameworkElement中增加Border和Padding的依赖属性；

并且将Border对象的属性修改为与css的border一致。

之所以采用dp，主要为了在模板之类的场景中进行设置。

把Background生成一个css的表达。写入到dom中。

这些元素

# Thickness

在WPF中Thickness是一个Struc类型的，不是DO，其几个属性都不是DP，因此这个地方的设计中，这些属性都是不能进行Binding，也没有值变化机制的。

对应的Margin是一个DP而存在的，在FrameworkElement上。

在WPF的设计中，Padding是存在于Border对象上的，并不是人人都有的一个属性。而在css中，BoxModel是css社会的基本规则，人人都要遵守，都具有天生遵守这个规则的基因。

在WPF中，如果不放置一个Border的话，就意味着没有padding了；

# 背景

渐变：线性渐变——从一个方向向另一个方向的线性变化，和径向渐变——水波纹一样的圆圈形式的渐变。

线性渐变的语法：angle|（boder|corder） colorStop +

ColorStop：color [length | percent]

径向渐变：background: radial-gradient(*center, shape size, start-color, ..., last-color*);

嗯，需要调整这两个对象，完全按照css3的形式来定义这两个对象。

ImageBrush、RadiusGradient、LinarGradient、SolidBrush等几种形式来改写这部分的对象。

为了css，这个部分的WPF内容，可能需要彻底地按照css来改造才行。

可以搞一个具有这么多个属性的background，从而使用一个对象就可以搞定，从而代替WPF的复杂设计。

只有这样，对于页面设计人员来说才是完备的，其缺陷是，无法由VS进行界面设计了。呵呵，这个决策会让人吐血的。不过，现在来看，还是采用css的要合理一些。

在XAML中，对于颜色属性，反正是一个字符串，对于字符串不能解析为规定格式的话，就试图用这个字符串查找预定义的颜色，如果不能找到，就是错误的。如果能够解析为规定格式的话，就使用这个规定格式好了。

在Summer中，反正color不会真用来自己绘制，当然可以采用字符串形式了。

之所以需要采用一个对象来表达背景，就是为了binding，换句话说，需要把背景相关的属性设计为DP，从而允许进行binding，这样才能体现出来方便性。但是如果这样的话，意味着在XAML部分，可以采用两种方式，一种是按照css的方式来进行简写，由编译器来自动创建对应的对象。

另一种当然可以通过对象元素的方式来进行，这两种方式都可行，才体现出来XAML的方便性。

不过可以通过一个转换器来实现把XAML转变成SUmmer的XAML，反之亦可以进行。

# Grid Layout

呵呵，要是GridLayout能够使用的话，WPF的grid布局就变得很容易了。

嗯，太可惜了，两外的主流浏览器都不支持，也看出来，将来什么时候能够支持。只好暂时先放弃了。真是太可惜了。尼玛，老子一生气帮这帮家伙来实现这个特性。

# Display：Table

这个css的手段看样子有可能让自己从过多表格的噩梦中醒来。

呵呵，这个有限制，就是对于table-cell设置中，缺少colspan和rowspan的特性，这条路也是走不通了，不过，但是可以用来实现StackPanel的水平方向布局。

在Stack的水平布局中，可以在Stack本身的div上使用display:table这个样式，然后，在每一个child元素上使用这个display:table-cell的样式，这样是否就可以进行水平布局了。原来的StackPanel布局会导致换行，当元素高度不一致时，会出现俄罗斯方块的堆叠效果。

Css的处理例外太多，input就不遵守display：table-cell的样式约束；居然这还是css2.1规范怂恿的。

这个样式表可能用户WPF的DataGrid中会更加合适。

或者在没有span的table中，会更加地合适，这样就可以优化一些掉一些元素了，例如：在Calendar中，那个日历的表格，就可以又花掉很多元素了。

WrappedPanel

# Input

这些表单元素也是遵守boxModel的，对于input type=“text”来说，如果内容很大，此时字体如果不匹配输入框的话，会显得很奇怪，所以这个地方就有个行高的说法。

如果把Input通过一个Div来包装的话，有点太大材小用了，可以控制需要时再添加这个元素，例如：在StackPanel中需要这个元素，那么就添加这个元素，不管可以认为这个元素上的发生事件，是StackPanel上的，在Input中只是记录而已，其逻辑关系并不改变；这样当删除一个input时，判断如果存在wrapper，就删除wrapper既可以了。

这样的话，就可以把Input

# Geolocation

嗯，这个特性三个浏览器都提供了支持；

# 服务器事件

IE11还不支持服务器事件特性；

# CSS3

其中有一个吧一个元素作为背景的定义，呵呵，快赶上WPF的思路了。

# 文件上传

在html5中，可以一次上传多张图片，传送成功后，会返会这些图片的url，意味着到了服务器端，就可以找到这些上传的图片了。当然需要一个专门的servlet，来接收上传的图片，然后管理url，一遍可以找到对应的图片。

# 可能的简化

1. 是否可以把EffectiveValues直接构造到dom上，减少二次存储；

EffectiveValues的存储可以直接利用dp的globalIndex来作为索引，直接保存到对象上，从而简化这个地方的查找；

# 焦点Style

在html中，采用focus伪类可以很容易的实现这个样式，但是，如果不采用伪类呢？

嗯，这个地方到底有无必要类似WPF的设计，每一个元素都要设置一个焦点style呢？还是可以在资源中设置一个全局的焦点样式，但是在WPF中，设置一个全局的焦点样式是一个很难的事情，没有办法为所有的元素一下子设置上样式。

如果通过baseon的话，其实是资源重复的；

如果为每一个元素定义一个focusStyle，呵呵，其实就是css的做法了。

但是，毕竟是一个解决方案啊。

在css中，这个焦点样式是由浏览器进行管理的，在WPF中，却是由自己进行管理的。这样会消耗资源地。这个地方好像业务伤大雅。

# ProgressBar

这个控件还没有实现；

# 1月22

今天继续思考向css的转换问题。

昨天的成果；采用display：table的形式来处理stackpanel的布局。

# Css3的Multi-columns

Css3中有一个多栏布局的样式，这个特性三个主流浏览器基本上都支持，对firefox，不能使用auto，这能说这个布局勉强能用

# 函数都是在prototype上吗

经过分析，这个地方应该是位于prototype上的。

# JSON

把一个对象的状态凝固下来变成一个字符串。

如何还原为原来的对象呢？

好比java的序列化一样，需要使用一个无参构造函数，可以在每一个类实例中添加一个Class有关的信息，然后就可以进行二次加工，进而变成对应calss的实例。

这个地方有几个问题：

1. 从类的角度看的静态方法，本来是没有必要进行序列化的，结果也没序列化了；

在dojo下面，一个对象实例的的结构，

可以如此实现：把json转换为js对象后，然后设置其prototype为类型即可吧。

# Javascript

Prototype：函数的对象的一个属性；

Constructor:可以通过new来执行的函数；

对于new来说，创建一个对象，设置其--proto--属性

使用这个对象调用构造函数；

反序列化：

增加一个特殊的构造函数处理，传递一个{}进来，然后依次把这个对象的属性设置到this上即可。

当然最好的办法还是通过-proto-

或者在每一个构造函数中都需要进行参数个数的判断。

# RPC部分

把进行了远过程标注的类，编译为对应的url，这个url进入生成的js，同时需要服务器上的进行部署的对应的。

编译为一段ajax调用的代码。

大概的形式，创建ajax，序列化参数，可以直接使用arguments，然后编译数组对象，通过url把方法调用参数发送到servlet，servlet解析参数，通过url解析出来对应的参数，然后进行方法调用。

呵呵，一个问题就是如何实例化要调用方法的对象呢？涉及到有状态和无状态两个策略。如果是有状态的，可能需要进行同步才行。无状态的情况吧。

可以创建一个实例，统一使用这个实例来进行调用。

可以通过固定的写法，类似于wS的形式，获取服务对象，然后调用这个方法。这个服务对象其实是一个存根，完成序列化，网络传输，并返回调用结果。

在XAML中编写这个部分的代码时，需要进行特殊的约定。

也就是说，需要定义一些接口，在这些接口上进行标注，然后还可以在类上标注实现了哪一个接口，呵呵，怎么感觉跟guice很相似呢？

在OSGI环境下，整合guice，然后在servlet中进行使用。从而完成接口和实现的灵活配置了。

这样这个Servlet就变得很简单了，找到实现的实例，然后通过反射的方式来调用这个方法，然后把结果进行json序列化，返回给客户端。

客户端拿到这个json后，进行统一的处理，此时为了能够兼容类型，就需要进行特殊的反序列化了。这个结果交给js调用的地方，进入WPF运行环境，例如：返回一个结果集合，把这个集合设置给ItemsControl，或者进行其他的设置。

# 编译环境

对于服务类，定义java接口，添加标注，这个标注主导编译，出来一个dsl格式的接口，这个接口应该是一个简化基本类型的。

通过dsl定义模型，其类型系统与java等价——嗯，WPF的集合部分的类有点棘手。

然后这个模型编译为两个：java类，XAML的类。

在dsl中有一个序列化json的方法，在java类中有一个反序列化json的方法。

甚至是自己定制的基于json的java序列化了。

# 应用演练——京东

通过默认页面到了浏览器后，各个部分开始通过ajax进行加载，由于这个是异步的，因此，会进行并发加载，逐步展现出来网页的内容，然后用户登陆，获取新的页面，在这个页面中。

看样子这个远程调用要先测试一下。

Ajax返回对象

# 1月23

已经三天了，在思考整体思路，呵呵，耽搁了局部的进展。

今天三天的学习思考，平台的卖点应该是html5的针对性。

采用WPF的设计思想，到底产生了什么好处呢？

1. 核心的好处是binding，为了这个binding，需要DP、Template、Style的配合，于是创造出了一些概念，作为binding强大的手段保障。
2. 把media、documents、form结合到一起了。理论上来说。

# 思想的强化

1. 可以把Canvas的API引入到XAML层中来，从而可以在dsl层面进行这个API对应的编程，来实现复杂的应用。
2. 把documents部分进一步与html5靠拢；
3. 把图形部分进一步与SVG部分靠拢；
4. 把WPF的很多属性进一步靠拢到css上；

# 集合部分

这个部分确实需要彻底修改，现在的实现没有充分利用js的优势和特性。

# 今日任务

## Validation：

先设想一下这个特性的实现，是在target更新source过程中产生的吧，对于source更新target来说，好像没有问题吧。

由于Source和Target之间的数据传递经过很多个步骤，因此每一个步骤都有可能产生错误，Validation就是需要针对这所有步骤的。

这个地方的设计分为如下的几个层面：

1. Validation的实施，完成数据验证，给出错误；
2. 错误的表达，binding过程中出现了错误，需要给出表达；如何表达这个出现的错误呢？假设就是设置一个特殊的style。这个设置应该是设置local，从而屏蔽了style中定义的同样的属性了吧。
3. 出现了错误后，source和target之间的数值交换如何进行；

忽然想到一问题，就是初始值是如何从source设置到target的呢？

SetBinding后，进入Attach、最后Activate，最后判断是否需要更新target（如果是onewaytosource的话，就不需要了），把source的value设置到expression上，然后调用DO的Invalidate（TargetDP）即可完成了。

## UpdateSourceTrigger.LostFocus是如何实现的呢？

在AttachOverride中，如果是UpdateSourceTrigger.LostFocus的话，就在对应的TextBox上挂接一个LostFocusHandler，然后在Expression的SetValue中，判断，如果不是UpdateSourceTrigger.PropertyChange的方式的话，就不会更新Source，从而实现了UpdateSourceTrigger.LostFocus了。

在Deattach中，需要把LostFocus的事件侦听Handler去掉。

在LostFocus中，只是调用Update（）即可，因为已经设置好了Expression的Value，在每一次的DP属性变化中，都设置了Expression的Value了。

上面的实现很自然，应该是可以设想到的。

## EffectiveValue

在DO中这个部分的代码实现，没有充分利用js的特性，如何优化一下呢？

先说说WPF的实现吧。

通过一个EffectiveValueEntry的类型，保存了具体的值，由于值得复杂性，因此特意是一个结构。在DO中设置了一个数组，自然这个地方就面临着一个问题，获取一个DP的值，是通过dp，dp上相互区分的是propertyIndex，这个是全局的唯一的。而操作这个数组的是另一个数组索引，这两个之间就不匹配了。为了这个不匹配，WPF采用了一些技巧，其核心就是js中关联表的一个实现，因此这个地方可以直接使用js的关联表，也就是直接使用propertyIndex来获取对应的EffectiveValueEntry好了。

如果进行上述优化，代码可以减少几百行，效率也应该会更高吧。

# MouseLeave

今天早上练拳想到一个解决方案：

就是在内部组成部分的dom上，设置标志，如果是这些dom发生mouseOut事件的话，检查这些标志，如果存在，可以忽略这个事件。这样的话，就可以比较好的解决了MouseEnter和MouseLeave事件了。

# 一个想法

就是在XAML中引入script的关键字，允许在合适的地方嵌入脚本，来模拟html中无处不在的<script> </script>脚本。

在XAML中引入Canvas的API，允许用户绘制Canvas上的图形。嗯，这个想法应该没有问题，只是需要把Canvas的WPI准确编译到XAML的编辑器中即可。

# MutationObserver

呵呵，好像忽然发现一个光明大道了。

就是这个MutationObserver，通过这个观察者，可以获取到dom树上的变化，当然包括属性变化。

有了这个机制后，就可以把WPF的属性直接利用dom上的值了，从而可以极大的减少effectiveValueEntry造成额外存储。

可能只是部分属性而已吧。对于style这个对象的子属性修改的话，应该不会产生变化事件了。嗯，确实是这样，否则style对象本身尤其复杂性，可能不好处理。

# CSS3的变换

呵呵，又是一大喜讯，三大浏览器都支持transform，例如：对一个Input的transform结果



嗯，真酷。

# CSS3动画

看看是否可以把WPF的动画用CSS3来实现。

先花些时间学习一下css3的动画吧。

Transition：property duration timing-function

Transition是把属性从当前值变成transition设置的值需要的时间。

设置一个动画，就是设置一个动画style，可以通过设置另一个style，来取消动画。

在WPF中，需要设置Animation的属性从开始到结束的值。

动画其实就是统一了一个named keyframes，然后在.animation {}中，通过name来引用已经定义的关键帧。

# Transition

呵呵，我明白了，原来这个过渡（感觉这个翻译比较好），在定义时，并没有涉及到具体的属性值。

应该是这么理解，style中设置了这个过渡后，当设置一个属性时，如果这个属性定义了过渡，就会按照这个过渡的定义来进行旧值到新值的过渡了。

这样的话，跟WPF好像不一致。

# WPF动画

通过Action或者Trigger来触发一个动画。

在css中，任何对属性的改变都会触发定义的动画。这个与WPF相比可能就是动画范围更加宽广了。

# VisualState

这个概念好像和css的伪类非常类似。呵呵，是否可以把WPF的上述概念变换为css的伪类呢？

# WebGL

三个主流浏览器都支持这个技术了，呵呵，也是一个令人激动人心的事情。

# 全屏API

这个也是三个主流浏览器都支持的；

# WebSocket

呵呵，三大浏览器也都支持这个技术了，呵呵，也是令人高兴的一件事情。

当然，这部分的API也需要体现在XAML的编辑环境中了。

# 1月24

把button、Check、Label等当成基本控件，不再允许存在Template（Control、Data），当然保留ContentControl，从而允许用户自己组合一些控件，达到灵活性的目的。

当然现在的ButtonBase等基类需要保留，以便可以实现复杂的组合控件。

现在的这些WPF实现的控件，可以统统在名称上添加ex，来区分。

当然基本控件上必须的属性都是DP的，一遍可以进行binding。

如果这样的话，基本控件既得到了极大的简化；

把HyperLink提取出来和Image一样，既可以是行元素、也可以是块元素，如果作为控件，就是块元素，如果作为文档中出现，就是行元素了。

# ContentEditable

在html5中，每一个元素都具有这个属性，用来可以表示，是否可以对浏览器元素进行编辑。嗯，这个特性很不错。

# ContextMenu

全局属性，允许为元素定义上下文菜单；没有一个浏览器支持，嗯，这个到可以使用WPF的实现了。

# Dragable

这个特性在三大浏览器上也都得以实现。是时候看看WPF的drag drop部分了。

DragStart

DragOver

DragEnter

DragLeave

Drop

DragEnd

在WPF中，需要AllowDrop进行设置，才会接收，也就是才会发生drop事件；

通过在mouseMove中判断是否按下了鼠标右键，如果右键按下，就进入拖动状态。

在Summer中，需要还原DataTransfer，以及相关的事件。

# HtmlEditor

# File Api

这个仅仅是接口定义。从什么地方开始呢？

从fileInput开始，

Input type=file

这个控件仅仅是选择了文件，当然可以支持拖放文件到其中，然后在form提交的时候，才会上传。

这个地方的一个场景，就是照片输入，

# Opacity

这个属性是css的标准属性；

# Visual3D

如果要支持Html5的3d的话，如何处理WPF的Visual3d呢？

# 爸爸账号

中国邮政储蓄银行

6210 984 73000 3790 111

曹淑文

# 大孃孃

1375 2199 8010 0110 613

中国建设银行江苏省分行南京市支行丹凤街分理处

张端明

# Transform

WPF中有两个属性：

在UIElement中RenderTransform、TransformOrigin

忽然才发现，原来，在css中，进行transform的function格式是很特殊，居然采用了函数的写法。

# 思路

需要把所有的思路重新梳理一遍

1. 界面
   1. 通过XAML编写，生成一个创建WPF控件的js脚本；应该是一个html和一个脚本。需要一个中间的DSL表示，可以表达定义的变量名称，可以参照其他类定义的Handler；

这个翻译如何进行呢？

XAML，之前已经初步实现了这个XAML编辑器了。还需要针对不同类型的属性，看看能不能给出提示，语法检查、超链接跳转等特性。

现在创建的这个DSL源文件，其实就是一堆标签而已，当然在内部已经处于半解析状态了。

形式就是dom。

也就是说一个XAML文件会有两个dsl文件，一个叫做view，另一个叫做sjs

这个view翻译为一个中间类，在这个类中，具有name就成为了类属性。

这个类就可以翻译为InitializeComponent方法，和一些私有属性。

这些属性在类上。

到了Behind class上，构造函数中调用InitializeComponent，当然也可以调用其它的方法了。或者说干脆这个构造函数就是固定的。通过Initialzie（）的方法，给用户来进行初始化工作；

其他的就是用户在这个类中编写EventHandler了。

这个地方的一个难点是，如何把这两个类在作用域上和成为一个类；这个对应的类是维持在内存中的，等到complier时，才会生生成js文件。

这个生成也是固定的位置的，跟包名称一起，在bin中；将来部署的时候，可以只是部署bin中的。

这个生成需要照顾dojo的配置，然后是require一堆，这个地方，需要考虑不同包名称但是，但是类名称一样的情况。

生成html，js可以单独生成，当然也可以在一起生成了。

这样就算完成了编译。

为了在类中使用这个dsl，需要把这个dsl变成一种类的形式。

Class XXX {

构造函数{

}

可以编写事件代码；

}，

当然可以在其他的Page中import一个Page了。

其实在语义检查阶段，都是采用的java语义，最终再把java语义编译为js脚本。

包括linq也是如此。

这个中间模型是通过infer来进行的。

# XAML编辑器

在这个编辑器里面，只有dsl相关的代码，也就说在这个地方并不能使用jdk的类，所有的类都是自己的。

这样在这个地方编写的代码就是完全可以编译为js的了。

于是剩下的问题的是，一个特殊的类，就是RPC的标注的类的。

这个类是需要同时以java和DSL的版本存在的。

标注中可以定义servlet吧，或者说请求路径。

还有就是业务模型类，这样的类也是具有特殊标注的。

这个地方的一个难点是，在dsl版本上需要进行类型简化才行。

在sjs的dsl中，语法尽可能与js一致，但是不能把prototype之类的js内部的信息泄露出来。

要折腾进来的API还是不少的。

模型反正需要一种特殊的dsl来编写。因此可以保证与js类型的兼容性，也好处理与java的兼容性。

Sjs使用服务接口，java需要提供实现。

这个地方的问题是数据的序列化和反序列化，以及异常等处理了。

呵呵，截止到今天，思路总算是清晰了，也算是确定了吧。

看看一共需要多少个dsl呢？

1. Entity（映射）——暂时不需要
2. Database（数据库）——暂时可以不管
3. View（XAML），需要在内存中
4. Sjs（javascript）
5. Model（业务模型）、和RPCService编辑器；这两个都需要同时编译为java和sjs两种类型。

在sjs中，jvmType都是自己的，在model中

上一个版本中的dsl设计中，有些地方的考虑还是比较肤浅，没有达到WPF的高度。

# 1月25

在XAML编辑器中，不再设想直接在grammar中，使用参照作为属性值。

如果这样的话，整个应该就简单了。

# Clip

修改clip为css的语义；一个矩形、auto、inherited

# Transform

这个属性如何可以看出来可以是一个级联的呢？这个地方通过一个TransformGroup的类来进行的。

# Animation

Transition

Animation，需要参照一个已经定义的，在动画定义中，通过关键帧技术，来定义几个关键帧。

# Tooltip

如果是简单的字符串，只是设置到title上即可。如果是其他的话，就按照WPF的实现来处理了。

# 继续回到WPF上

Expander实现了.

呵呵，又发现一个问题，比较讨厌的问题。

在Expander中，其ControlTemplate中有一个ToggleButton的子控件，这个子控件的IsChecked和Expander的IsExpanded属性双向binding到了一起。这个binding是定义在模板中的。

在模板展开时，碰到了问题，找到了Expander，按照IsExpanded和Expander这个类型来查找DP时，由于此时IsExpanded还没有加载——使用的懒加载技术，于是Binding就进行不下去了。

几个解决办法，一个是把懒加载技术变成静态初始化，另一个是在需要时保证要加载。

先采用后者吧，在解析的地方，先加载一下看看。

嗯，还需要花一些时间看看模板中的binding后的问题。

如果是RelativeSource的binding的话，此时target如何更新source呢？

# 小成果

Expander终于有点可以折叠和收起了，可是自己却高兴不起来。

因为是采用了比较别扭的办法。

Expander的ControlTemplate是采用Height为期望尺寸或者是为0，从而实现折叠和展开的。

在Html中，tr的Height设置为0，其效果并不是预期的，虽然高度成为了0，但是其仍然会显示td的内容。设置tr的高度，只能是在td的内容小于这个高度时起作用，如果放不下，会自动撑篙tr的。

Div的高度为题，如果设置为0px的话，也是不会成为真正的0px的，如果此时有内容的话，其高度是内容的。

并且input之类的，即使设置其高度为0，也是会有一个小方块出现，虽然很小。

这个地方与WPF的区别很大，WPF是可以通过设置一个元素的高度为零，从而隐藏一个元素的。

只是不知道，对于高度为0的元素，其焦点如何处理？

高度为0，hitTest检测不到，同时在焦点移动计算中，也是会排除高度为0的元素，因此WPF好像没有问题。

# 1月26

完成树的折叠吧。

折叠逻辑上完成了，也就说作用到了需要折叠的地方，但是有问题——由于不是WPF。这个地方设置ItemsHost的div为display：none，但是由于tree是建构在表格上的，因此对应的tr还在。问题也是比较讨厌的。

并且这个地方是修改了UIElement的OnVisibilityChanged，在这个方法中，设置dom的style的display：none。这个也是有问题的，有时候，我们需要display：inline-block之类的信息，会被覆盖，导致问题。

这个问题先放着。

# ItemsControl的Select

先看看代码中关于选择的实现机制吧。

这个地方的核心还是CollectionView有关吧。

这个地方的概念有：CurrentItem、SelectedItem、FocusItem之类的。

ColectionView表达了CurrentItem、以及Item的Navigation；并没有涉及Select、Focus之类。

从代码来看，这个Select机制还是比较复杂的，到底为什么要这么复杂呢？

几个类：

SelectionChanger

ChangeInfo

InternalSelectedItemsStorage

Extended的选择模式：

当 SelectionMode 属性设置为 **SelectionMode.MultiExtended** 时，按下 Shift 键的同时单击鼠标或者同时 Shift 键和箭头键之一（向上键、向下键、向左键和向右键），会将选定内容从前一选定项扩展到当前项。 按 Ctrl 键的同时单击鼠标将选择或撤消选择列表中的某项。 当该属性设置为 **SelectionMode.MultiSimple** 时，鼠标单击或按空格键将选择或撤消选择列表中的某项。

# TabControl

页签是一个ItemsControl，只是显示的是Header的内容而已。

页签内容只是显示一个选择的哪一个Item而已。

这个地方在页签点击后，设置focus，然后在PreviewGetFocus中，处理Tabitem的选择问题。

# 1月27

可以在Number的基础上，实现其他的数字类型，但是是否还支持数字相加之类的呢？

这个问题先放一放吧。

在TabControl中，当当前项发生了变化后，页签内容是如何响应的进行变化的。

# 新问题

就是如何删除已经创建的元素。

例如：在Template变化后，需要删除原来的内容，添加新的内容。呵呵，总是会出现新问题。

解决办法：既然会删除元素，本质上调用RemoveVisualChild（）的方法，在这个方法中，在从父元素伤处时，可以同时处理dom之间的关联。

对于Documents部分的元素的删除有些复杂，WPF中由于RickText也是同样的技术实现，因此就会涉及到UndoManager部分的处理，加上整个元素组织成一棵树，既具有文档的视角，也具有元素的视角，因此比较复杂。

在Summer中，既然是有专门的集合，那么在集合的添加、删除中，可以同步维护dom文档的关联。

# 1月28

选择部分思路：

在SelectionChnager类中，维护临时选择和取消选择的集合；在Begin、End中启动和结束真正的选择。

在Selector中维持一个集合，用于记录当前选择的项目（如果source变化的话，需要维持一致才行）；

在有关的SelectedIndex上，也是与Selector上的选择集合有关（获取第一个元素的index）；

和CollectionView的关系——是ItemsControl的核心属性。CollectionView完成代表当前项含义的属性、实现对数据集合的导航。处理ItemsSource和控件集合之间的协调；

CollectionView：Current、Sort、Group、Filter；

# ItemsControl

通过一个集合ItemCollection来管理自己的数据，分为两种模式：Direct、ItemsSource，在Direct Mode中，内部是创建了InnerItemsCollection来管理自己的逻辑集合；

另一个是通过ItemsSource来管理自己的逻辑集合，这两个模式是互斥的，并且一旦进入一个模式，需要清除后，才能在进入另一个模式；

对于通过ItemsSource的属性设置的集合会变成某种CollectionView（ListCollectionView、BindingCollectionView），对于ObservableCollection，会自动侦听集合的内容的变化。

为了设置Group和Sort之类，又搞出来了以个CollectionViewSource这个一个对象，专门用来实现binding。

# TabControl

糊里糊涂，TabControl选中内容部分就可以显示当前选中的了。

在TabControl中设置一个Person的集合。

为Person制作了DataTemplate，这个Binding中，使用的是DataContext属性。

现在当我们导航CollectionView时，页签的内容随着变化。

但是是什么机制呢？

分析一下先：

在TabControl中，SelectedContent是一个DP，在ControlTemplate中，对ContentPresenter的内容设置是SelectedContent，并且是TemplateBinding的，也就是说，当TabControl的SelectedContent发生变化后，ContentPresenter是能够感知的。

只是在ContentPresenter中对内容的变化是如何处理呢？

在ContentPresenter的OnContentChanged中，设置了DataContext，嗯，现在的现象可以解释了。此时ContentPresenter，没有变化，由于其内容是通过Binding到DataContext的，因此，修改了DataContext后，此时ContentPresenter的内容自然变化了，这个变化没有更换模板，也没有创建新的控件来表达内容。

只有更换ContentPresenter才会引起ContentPresenter内容的变化。

如果由于当前项的改变，造成 选择的改变，在TabControl的UpdateSelectedContent中，就会把当前的TabItem的属性设置到Control上，设置到TabControl上，由于ContentPresenter是通过TemplateBinding来Binding二者的，因此就会引起ContentPresenter的Content、Template之类的属性变化，从而改变ContentPresenter的内容了。

# 新问题

在TabControl中，可以通过ItemTemplate和ContentTemplate来处理Header和面板内容的模板了。当然可以通过style也是可以的，可能style的效率稍微低一些吧。

现在问题是：在TabItem中设置了Trigger，当IsSelected时，修改TabItem的背景，但是这个IsSelected好像修改了两次，第一次被设置为true，接着又被改为false，导致背景不能正确设置了。

问题找到了，虽然隐藏的比较深。

在SelectedItemCollection中，有两个方法，Add(ItemInfo into), Remove(ItemInfo info)，由于SelectedItemCollection是从ObservableCollection继承的，也有这样两个方法，于是子类的这两个方法，就覆盖了父类的方法，导致被选择了两次，于是IsSelected就从true又变成了false，从而导致背景丢失。

根据参数判断这两个方法，来决定要执行哪一个版本的方法，从而解决了问题。

以前修改了SelectedItemCollection的私有属性\_updatingSelectedItems的默认值为true，导致又出现了问题。还好，现在把它该回去了。

# 1月29

TabControl已经研究了好几天了，修正了一些bug，应该会运行的更加流畅了吧。

接下来的问题：

页签控件的初始选择为什么没有导致触发器修改了自己的背景呢？

在SetCollectionView（ItemCollection）中，有机会设置TabItem的IsSelected，但是由于此时TabItem还没有创建出来，因此也就没有设置。

第二个地方，是在创建出来TabItem集合后，是否需要同步TabItem和CollectionView中的当前项呢？这个地方应该是在Panel中。

# Style先处理掉

还是直接把ForegroundProperty的类型改为color吧；

Border是其中的关键，里面有Padding、Border等属性的处理。

CSS3的Border属性：

# Background

background:url(/i/bg\_flower.gif);

background-size:33px 50px;

background-repeat:repeat;

|  |  |
| --- | --- |
| repeat | 默认。背景图像将在垂直方向和水平方向重复。 |
| repeat-x | 背景图像将在水平方向重复。 |
| repeat-y | 背景图像将在垂直方向重复。 |
| no-repeat | 背景图像将仅显示一次。 |
| inherit | 规定应该从父元素继承 background-repeat 属性的设置。 |
| space | 应用同等数量的空白到图片之间，直到填满整个元素 |
| round | 缩小图片直到正好平铺满元素 |

background-origin（border | padding | content）

background-position:

background-attachment:

|  |  |
| --- | --- |
| **值** | **描述** |
| scroll | 默认值。背景图像会随着页面其余部分的滚动而移动。 |
| fixed | 当页面的其余部分滚动时，背景图像不会移动。 |
| inherit | 规定应该从父元素继承 background-attachment 属性的设置。 |
| local | 。这是用来配合可以滚动的元素的(设置为 overflow: scroll; 的元素)。当 background-attachment 设置为滚动(scroll)时，背景图不会随元素内容的滚动而滚动。 |

background-clip

|  |  |  |
| --- | --- | --- |
| **值** | **描述** | **测试** |
| border-box | 背景被裁剪到边框盒。 | [测试](http://www.w3school.com.cn/tiy/c.asp?f=css_background-clip) |
| padding-box | 背景被裁剪到内边距框。 | [测试](http://www.w3school.com.cn/tiy/c.asp?f=css_background-clip&p=2) |
| content-box | 背景被裁剪到内容框。 | [测试](http://www.w3school.com.cn/tiy/c.asp?f=css_background-clip&p=3) |

background-image: url(top-image.jpg), url(middle-image.jpg), url(bottom-image.jpg);多背景

background-break

\* Background-break: continuous; 默认值。忽略盒之间的距离(也就是像元素没有分成多个盒子，依然是一个整体一样)  
 \* Background-break: bounding-box; 把盒之间的距离计算在内  
 \* Background-break: each-box; 为每个盒子单独重绘背景

# CSS处理

在SUmmer中，css的长度可以采用字符串表示，或者说css的属性都采用字符串表示，只是在XAML编辑器中，需要进行有效性检查。这样的话，就尽可能地兼容css了。

如果采用数字之类的话，会存在单位之类的问题，不好处理。

这个地方需要与WPF分道扬镳了。

# Gradient

**<linear-gradient> = linear-gradient(**

**[ [ <angle> | to <side-or-corner> ] ,]?**

**<color-stop>[, <color-stop>]+**

**)**

**<side-or-corner> = [left | right] || [top | bottom]**

**这个直接使用一个字符串属性来表达是否合适呢？**

**这样就不再需要WPF过于复杂的那么一大堆对象了。**

**<radial-gradient> = radial-gradient(**

**[ [ circle || <length> ] [ at <position> ]? , |**

**[ ellipse || [ <length> | <percentage> ]{2} ] [ at <position> ]? , |**

**[ [ circle | ellipse ] || <extent-keyword> ] [ at <position> ]? , |**

**at <position> ,**

**]?**

**<color-stop> [ , <color-stop> ]+**

**)**

**<extent-keyword> = closest-corner | closest-side | farthest-corner | farthest-side**

# Transition

# Transform

# Animation

# CSS

Css中的class概念能够使用吗？

如果这样意味着让开发人员彻底地使用css的概念来进行开发了。

如果把外观相关的属性都采用String类型的话，有没有问题呢？

在WPF中，有些属性的类型是Animation——为了让这个属性支持动画，例如：Brush，之所以这样设计，有一个场景需要考虑，那就是对于SolidBrush来说，当颜色发生变化后，能够作用到控件上，这个作用是通过属性变化通知来实现的。

在WPF中，此时Brush是一个字符串的话，就不会作用到控件的外观了。因为需要通过属性变化来产生InvalidateMeasure或者InvalidateArrage来显示外观改变。对于Width、Height之类都是如此。

呵呵，从这个角度看，WPF如果充分利用浏览器的特性，应该是可以得到很到的简化的。呵呵，可能要去掉一多半了。

真正核心的是Template和Binding、DP、Command、RoutedEvent等概念了。

Animation、Transform等依靠Css来实现了。

这些属性的继承也可以不再需要了，只有DataContext之类的属性需要，而这个是业务需要的，不再是纯粹的显示需要了。

如果要处理属性变化的问题。

例如：修改字体，此时，就可以直接把数字改为xxxpx然后设置到style上即可。此时这个地方

# WPF显示有关的属性

Box-Model

Width、MinWidth、MaxWidth

Height、MinHeight、MaxHeight

Border（圆角、Image、Widht、Color、Style等）

Padding

Margin

Foreground：对应到css的color，用来表达显示内容的字的颜色；

Background（呵呵，一大堆的属性）

Transform（Rotate、Translate、Skew、Matrix）

Transition

Animation

KeyFrames

Font

Weight

Face-Name

Stretch

Style

可以把Font当成一个对象吧，这样是否会好一些呢？

Outline

* outline-color
* outline-style
* outline-width

为了简化处理WPF的KeyboardNavigation，可以增加属性，对几个箭头键的导航进行设置。

# 键盘导航

# 1月30

今天是年30，呵呵也是一年马上过去了，新的一年来了。

爷爷、奶奶通过视频看了天天，天天大声叫奶奶，奶奶高兴的皱纹都堆到一起了。

今年的春晚是冯小刚导演的，还有一点新意，只是看了大半段而已，后来就要哄天天睡觉了；

岳父岳母忙着打麻将，姨妈、姨父都在，12点之后，姨父姨妈回家；

呵呵，今年的除夕，小区东门不让放鞭炮了，基本上只是零零星星的，一段时间以来，西安雾霾很严重，让大家深切的感觉是生活在同一个天空下，因此放鞭炮的人也少了很多。

晚上给老朋友们发了贺年短信，花了1个半小时，看着老朋友的回复，是很开心的一件事情；

# Dom元素的不可见

是否可以通过暂时从domcument中删除当前元素的方式来实现可见和不可见呢。

现在的问题是，比较不好处理的。

例如：一个div被设置为display:table，如果隐藏的话，需要设置display：none，但是一旦显示的话，就需要重新使用回display：table才对，呵呵，这个还真是一个问题。

可以针对每一种控件，记录下来原来的display：的属性，如果变成显示的话，还可以再次显示回来。

嗯，visiblity为hide的话，确实是仅仅让元素不可见，但是元素还占用了对应的空间，只是这个空间为空的。

设置元素的高度为0px的话，并不会隐藏元素，看样子元素是按照lineheight来处理的。

设置为width=0px也是不能让元素不可见。

此时的boxmodel确实是0，但是其内容却显示出来了。

此时元素的hittest是按照时间内容的范围来测试的。

# Summer

经过一年的探索，Summer的思路越来越清晰了；

借用WPF的思想、依靠Html5、CSS3，实现一个丰富、强大的web平台。

融合表单、Documents、Media、2d、3D的表现系统。

在布局方面，冒一下险，充分使用css3的布局元素，不再使用表格来处理。

这样的话，WPF的Grid布局，基本上可以完整地迁移到web上，

StackPanel采用FlexBox

WrapPanel采用Column Layout

DockPanel采用Grid来实现吗？

这样在页面上，就不会在出现布局需要的元素，dom和控件之间基本上就是1对1的关系了（除了SVG），这样的话，对于事件的处理——尤其是mouse的over、out、leave就容易多了。

表单元素充分引入html的；

动画、过渡、变换直接使用css，如果采用wpf的手法，会损失css3的特性。

接下来需要看看documents的手法了。在html5中引入

# text-overflow:ellipsis

呵呵，这个属性还是很有意思的，很多地方都可以使用了。避免了自己计算忽略的麻烦。

# 新思路

可能需要调整为html的表达方式才好，只是引入即可布局元素来处理布局，其他都可以直接采用html的原生做法。这样的话，就容易处理多了。

当然，这样的html编辑器，还是有点区别的，因为可以进行语法检查，和支持交叉参考，并且这个交叉参考是结合了类的。

当然，如果能够不全部再用客户端js就更好了。

理论上应该是可以的，在对dom初始化后，建立一层对dom的封装，并且在这个过程中，建立起运行时实例的交叉参考。从而就可以是静态的页面了。

如果能够严格地让dom和Control之间建立起1：1的关系，这个地方也是可以得。因为可以使用dom元素来关联彼此。嗯，有了这个思路，又可以向前推进一大步了。

当然没有必要直接是让html作为根，还是需要一个逻辑概念Page来表达比较好。

如果能够保证严格的1：1的话，路由事件就可以采用dom本身的了吧。

# Documents

WPF中，这个部分比较html，感觉不够流畅，看样子这个地方需要采用html的表达方式才好。

几个问题，如何表示分栏内容；

浮动如何处理；

InlineUI、BlockUI之类的概念很不自然，其实只是部分模拟了浏览器的特性而已。

在现在的设计中，也存在floater的困局，还是类似于float之类的，需要一个对象来表达floater这个概念呢？

# 调试

可以把程序调试转移到IE上来，看样子其调试器也是不错的。

# Hidden

元素上有新属性hidden，来实现元素的隐藏，呵呵，终于比collpase好用了。

# 新思路

Page中可以是任何一个元素，可以选择是静态还是动态的形式。

静态就是可以编译为静态html，动态就是编译为js和html；

静态就是是仅仅生成html元素，和css。动态是添加了脚本；

这样的话，甚至可以把一个StackPanel设置一个float属性，从而嵌入文本中了。呵呵，应该不错。

Binding部分有无问题呢？因为Binding是建立在DependencyObject上的，对于ContentElement和Visual来说应该是没有问题的。可能会对逻辑树和VisualTree有影响吧。

每一个元素都可以有自己的孩子，分别针对FrameworkElement、ContentElement有所区别而已。

也就是Page中，允许是ContentElement或者是Visual。

很快一个问题出现了，就是在StackPanel中是否允许其他元素呢——例如：ContentElement？

对于一个Control来说，默认是可能是Block、或者Inline，可以通过设置Display：Block或者Inlie来变成另外一种布局类型。

假设

<page>

<body>

<p>

</p>

</body>

</page>

# 1月31

今天是马年春节。

带儿子去了万达广场，做了电马和摇摆椅，儿子很高兴。

Summer的设计思路在html的影响下，受到了很到的触动，可能要按照Html的思路修正了。

对于一个Page来说，其根元素完全可以是任何一个Html元素。

元素之间的组合按照html的规则进行。

增加了几个几个控件，当然这些控件首先是块级控件。或者说基类是html元素，这样就可以和其他元素一样插入到任何能够出现的地方了。

是否允许在html元素上使用模板呢？

应该没有必要，可以把这些html元素看作最基层的构成材料。

看样子最好是都从FrameworkElement开始。

FrameworkElement也是Shape的基类。

如果也是Html元素的基类的话，应该没有问题。

WPF的问题在根子上，把Document表达在Form上，而不是像html那样把form表达在douments上。

如果采用html的dom来处理路由事件会有什么问题呢？

如果让Summer的对象这一层不再表达之间的关系，而是依靠dom来联系，那么就可以不必是纯粹的js来创建dom了。

也就是可以去掉WPF的Animation、Documents、以及布局计算有关的代码了。

接下来先采用css3来实现Control的布局吧。

Template的作用到底是什么呢？

表达特定控件的，或者说是要自己自己创建一些复合控件。

基本控件：类似于Border、Input之类的都可以认为是基本控件，但是为了扩展，允许有自己的ControlTemplate了；

所有的元素都是可以通过FrameworkElementFactory来创建的。

一个困难首先是元素的孩子问题。

还是可以分为逻辑和VisualTree两个组成部分，这个应该没有问题。

对ItemsControl的需求，封装一个控件还是还有必要的。

# CSS3布局

Flex Box布局

分为容器和Item，也就是Flex Container、Flex Item

对于容器需要：display:flex

还需要设置两个轴的一些特性。

# 父亲生日：阴历7月初九；

母亲生日：阴历正月十六，改为每年的正月初二过；

# Grid Layout

如果采用Grid Layout的话，grid就变得非常容易了，甚至都不需要这个布局了，当然为了简化，还是需要进行简单的封装——主要为了ItemsControl中使用。

Rowspsan和ColumnSpan就变得很容易了。

不用采用什么复杂的算法来进行计算，只需要把css的属性设置好即可。

DockPanel，这个布局就不需要了吧。加入真的需要，就自己从grid中搞一个出来好了，这样简化，反而增加了灵活性。

StackPanel采用Flex Box、Grid采用GridLayout，基本上就搞定了。

对于Tab的几个方向，可以通过FlexBox来实现吧，或者自己通过grid来实现。呵呵，这样还是很灵活的啊。

如果采用gridLayout的话，如果像设置某一行的背景颜色的话，就没有那么容易了。只能找出其row为某个值得那些元素，依次设置这些元素的背景了。

在目前的应用中，还没有需要进行正行选择的场景，对于ListBox，如果其中有多个字段，其实还是一个ListBoxIten而已，并没有整行选择问题。

[grid-column-align](http://www.w3.org/TR/2012/WD-css3-grid-layout-20120322/#grid-column-align)’ and ‘[grid-row-align](http://www.w3.org/TR/2012/WD-css3-grid-layout-20120322/#grid-row-align)’

在GridLayout中，如果设置了Item的Width和Height的话，意味着，此时span不再起作用。而是按照设置的宽度和高度进行了。

呵呵，一个伪问题，居然花了一个小时的时间。

在制作GridLayout的demo时，发现单元格之间居然留有很大的空白，找遍规范，也没有能够找到消除这个空白的属性，但是在另一个例子中，发现竟然没有这个空白，一直是怀疑有什么属性设置自己没有找到，知道最后才发现，原来那个空白是空行和空列，呵呵，真是一个笑话。

不过，这一通折腾也没有白费，算是基本上掌握了GridLayout了。

这个布局就比较起来Table，就像是一个盖墙来间隔彼此，一个是凭借图纸，也不能说凭借图纸就一定比间隔一到真实的墙又优势，但是毕竟还是解决了很多墙间隔带来的问题。

几个概念

定义grid

定义row和column，这个地方在规范中，row和column有几种定义方式，named、anonyous、template等，定义行列的关键是定义行高和列宽，以便形成单元格。

在元素上选择自己所在的单元格，选择自己的对齐方式，选择单元格跨越的行列。

至此一个grid算是基本上完成了。

这个地方有几个实验结果：

1. 如果元素没有定义width和height，自动使用单元格的高度和宽度；
2. 如果定义了width和height，则使用自己的高度和宽度，即使横跨了行和列；
3. 几个元素位于一个单元格内，遮挡部分是按照zindex的数值呈现的；
4. 几个元素位于一个单元格内，各自的跨行跨列各自独立存在；
5. 比较WPF，没有一个类似于显示表格线的开关；
6. 如果行列跨度设置为auto的话，意味着单元格的宽度有自身的实际宽度决定；
7. 如果没有定义col和row在grid上的话，该如何呢？
8. 试验一下，定义了grid，之后如果没有定义行列，此时的子元素如何编排呢？
9. 如果元素定义了row、column属性的话，如何呢？

此时好像会自动出现各自来对应row、coumn属性指定的单元格。

1. 如果没有为grid的子元素指定row和column属性的话，有无默认值呢？

没有看到默认值，但是好像会自动唯一row=1，col=1的单元格里。

从某种意义上来看，不知道是css借鉴了WPF的grid的手法，还是WPF的grid借鉴了css的gridlayout手法，反正微软在这个规范上我有先行者的优势。

从这个意义上来看，gridlayout比较flex box布局就简单一些了。

# 系统资源

也可以设置一个系统资源，把控件的模板定义在这个系统资源中，当成兜底的资源，当然用户可以在自己的页面资源中定义自己的控件模板了。

# 蛇年小结

今年最大的收获是终于确定了Sumemer的设计思路

年初研究LINQ

4月份确定了基于Xtext的开发思路，但是很快被XAML的出现否决了；

5月份尝试XAML，基本上可行；呵呵，平生第一次去了检察院，作为藏跟林案子的证人，感叹——为人莫做亏心事，半夜敲门心不惊。

6月份开始全力以赴研究WPF的实现

7月份开始js的迁移，老婆儿子回了西安；

8月份迁移了很多，10好开始练拳

9月份继续迁移，开始喘不过气来，在祈福医院检查了，回到西安后到西京医院进行了很多检查，最后实在蔡云的中医科，按照气滞吃了很多中药；断断续续持续到今年；第一次深切的感受到死亡与活着困扰。

10月份、11月份继续迁移、12、1月份

一大收获是基本上搞明白了XAML、WPF的实现原理。

对js、dojo熟悉了；

到了14年2月份，基本上确定了summer的方向是html5、css3，呵呵，到了这会儿，思路进一步明确了。

# 2月1

先把gridlayout实现了吧；

现在的grid中，采用了集合来管理row和column的定义，其实在css下，这个是没有必要的，都是一个字符串而已，因此，可以把column和row定义变成两个dp，类型为字符串就可以了。

在css中，已经处理了没有定义grid的col、row的情况，因此在容器中，就是没有定义也没有关系了。

而不再需要WPF的处理手法，因为WPF是需要自己来处理这个布局，因此就需要进一步明确的定义了。

如果这样的话，就可以减少了5个类，DefinitionBase、ColumnDefinition、RowDefinition、ColumnDefinitionCollection、RowDefinitionCollection、当然grid本身也得到了极大的简化，甚至这个类都没有存在的必要了，呵呵，先不要这么极端，暂时保留吧，为了可以兼容万一今年出不来的GridLayout的firefox和chrome；

去掉了这个之后，XAML中的定义也变得非常简单了。因为这个地方仅仅是一个字符串而已了。

Grid的移植算是基本上完成了。

TreeView也算是比较好看了。

在TreeView的解决过程中，顺便解决了一个隐藏很深的问题。

就是在ItemStructMap类中，在EnsureEntry方法中的代码导致了数据的混乱。差不多花了大半天的时间才找出来，最后还是通过采用js Array的splice方法来实现的数组插入。

接下来要完成：StackPanel，采用Flex Box，基本上实现了Flex下的StackPanel。

明天测试ComboBox、Calendar这两个控件，采用新的布局方案，其效果如何。

在TabControl中，可以为选中元素的ContentPrsenter设置一个Padding，让其背景和选中页签的背景一致，应该基本上就可以表达选中页签的效果了。

在现在的体系下，基本上保证了Control和Dom元素之间是1：1的关系。

因此事件应该比较容易处理了。

那就把事件实现了吧。

# 姨父姨妈回家

上午包饺子，他们没有回来吃，下午才来，吃完饭，饭后打了麻将；

# 动画

这个地方在keyframes上有点特别，就是这个地方需要一个名称，而这个是没有办法定义在行内的style上的吧。

倒是可以作为一种资源定义在ResourceDictionary中。然后在Animation的name属性上使用这个命名的keyframes；

# Css

Top、left等属性，用来设置dom元素的位置的。

也就是说，一个CssRule的规则，还是需要css的属性才行，呵呵，这样就流畅了。

# 明日任务

先下决心把background、margin、padding、border、Transform、Transition、Animation等实现吧。

把这些属性都定义到FrameworkElement中好了。

可以按照名称分开，还是定义一个对象，来表达呢？先统统定义到FrameworkElement中吧。

为了提高效率，可以通过值变化事件，来实现拼凑dom上的style。这样可能就比较有效率了。

如果是通过值变化事件来构造style的话，就不用担心附加属性的问题了。

还是把属性按照其功用分解到不同的类中，例如：background、border等；

至于marging、padding可以直接定义在frameworElement上了。

还有文本属性相关的附加属性；

通过附加属性组织到一起后，将来有可能变成一个可以实例化的类，在FrameworkElemeng中使用这个类即可。可能会让代码更加优雅一些，但是需要付出InheritanceContext的代价。

# Background

* background-color
* background-position
* background-size
* background-repeat
* background-origin
* background-clip
* background-attachment
* background-image

# Border

**CSS 边框属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| [border](http://www.w3school.com.cn/css/pr_border.asp) | 简写属性，用于把针对四个边的属性设置在一个声明。 |
| [border-style](http://www.w3school.com.cn/css/pr_border-style.asp) | 用于设置元素所有边框的样式，或者单独地为各边设置边框样式。 |
| [border-width](http://www.w3school.com.cn/css/pr_border-width.asp) | 简写属性，用于为元素的所有边框设置宽度，或者单独地为各边边框设置宽度。 |
| [border-color](http://www.w3school.com.cn/css/pr_border-color.asp) | 简写属性，设置元素的所有边框中可见部分的颜色，或为 4 个边分别设置颜色。 |
| [border-bottom](http://www.w3school.com.cn/css/pr_border-bottom.asp) | 简写属性，用于把下边框的所有属性设置到一个声明中。 |
| [border-bottom-color](http://www.w3school.com.cn/css/pr_border-bottom_color.asp) | 设置元素的下边框的颜色。 |
| [border-bottom-style](http://www.w3school.com.cn/css/pr_border-bottom_style.asp) | 设置元素的下边框的样式。 |
| [border-bottom-width](http://www.w3school.com.cn/css/pr_border-bottom_width.asp) | 设置元素的下边框的宽度。 |
| [border-left](http://www.w3school.com.cn/css/pr_border-left.asp) | 简写属性，用于把左边框的所有属性设置到一个声明中。 |
| [border-left-color](http://www.w3school.com.cn/css/pr_border-left_color.asp) | 设置元素的左边框的颜色。 |
| [border-left-style](http://www.w3school.com.cn/css/pr_border-left_style.asp) | 设置元素的左边框的样式。 |
| [border-left-width](http://www.w3school.com.cn/css/pr_border-left_width.asp) | 设置元素的左边框的宽度。 |
| [border-right](http://www.w3school.com.cn/css/pr_border-right.asp) | 简写属性，用于把右边框的所有属性设置到一个声明中。 |
| [border-right-color](http://www.w3school.com.cn/css/pr_border-right_color.asp) | 设置元素的右边框的颜色。 |
| [border-right-style](http://www.w3school.com.cn/css/pr_border-right_style.asp) | 设置元素的右边框的样式。 |
| [border-right-width](http://www.w3school.com.cn/css/pr_border-right_width.asp) | 设置元素的右边框的宽度。 |
| [border-top](http://www.w3school.com.cn/css/pr_border-top.asp) | 简写属性，用于把上边框的所有属性设置到一个声明中。 |
| [border-top-color](http://www.w3school.com.cn/css/pr_border-top_color.asp) | 设置元素的上边框的颜色。 |
| [border-top-style](http://www.w3school.com.cn/css/pr_border-top_style.asp) | 设置元素的上边框的样式。 |
| [border-top-width](http://www.w3school.com.cn/css/pr_border-top_width.asp) | 设置元素的上边框的宽度。 |

* border-radius

border-top-left-radius:2em;

border-top-right-radius:2em;

border-bottom-right-radius:2em;

border-bottom-left-radius:2em;

* box-shadow
* border-image
* border-image-source
* border-image-slice
* border-image-width
* border-image-outset
* border-image-repeat

# Transform

就是一个属性，可以写复杂的内容而已，好像不需要分解，分解并没有太大的好处；

涉及到变换的串联写法，更不好分解了。

# Transition

* [The ‘transition-property’ Property](http://www.w3.org/TR/css3-transitions/#transition-property-property)
* [2.2.The ‘transition-duration’ Property](http://www.w3.org/TR/css3-transitions/#transition-duration-property)
* [2.3.The ‘transition-timing-function’ Property](http://www.w3.org/TR/css3-transitions/#transition-timing-function-property)
* [2.4.The ‘transition-delay’ Property](http://www.w3.org/TR/css3-transitions/#transition-delay-property)
* [2.5.The ‘transition’ Shorthand Property](http://www.w3.org/TR/css3-transitions/#transition-shorthand-property)

# Layout 属性

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **属性** | **描述** | **IE** | **F** | **O** | **W3C** |
| [clear](http://www.w3school.com.cn/htmldom/prop_style_clear.asp) | 设置在元素的哪边不允许其他的浮动元素 | 4 | 1 | 9 | Yes |
| [clip](http://www.w3school.com.cn/htmldom/prop_style_clip.asp) | 设置元素的形状 | 4 | 1 | 9 | Yes |
| [content](http://www.w3school.com.cn/htmldom/prop_style_content.asp) | 设置元信息 | 5M | 1 |  | Yes |
| counterIncrement | 设置其后是正数的计数器名称的列表。其中整数指示每当元素出现时计数器的增量。默认是1。 | 5M | 1 |  | Yes |
| counterReset | 设置其后是正数的计数器名称的列表。其中整数指示每当元素出现时计数器被设置的值。默认是0。 | 5M | 1 |  | Yes |
| [cssFloat](http://www.w3school.com.cn/htmldom/prop_style_cssfloat.asp) | 设置图像或文本将出现（浮动）在另一元素中的何处。 | 5M | 1 | 9 | Yes |
| [cursor](http://www.w3school.com.cn/htmldom/prop_style_cursor.asp) | 设置显示的指针类型 | 4 | 1 | 9 | Yes |
| [direction](http://www.w3school.com.cn/htmldom/prop_style_direction.asp) | 设置元素的文本方向 | 5 | 1 | 9 | Yes |
| [display](http://www.w3school.com.cn/htmldom/prop_style_display.asp) | 设置元素如何被显示 | 4 | 1 | 9 | Yes |
| [height](http://www.w3school.com.cn/htmldom/prop_style_height.asp) | 设置元素的高度 | 4 | 1 | 9 | Yes |
| markerOffset | 设置marker box的principal box距离其最近的边框边缘的距离 | 5M | 1 |  | Yes |
| marks | 设置是否cross marks或crop marks应仅仅被呈现于page box边缘之外 | 5M | 1 |  | Yes |
| [maxHeight](http://www.w3school.com.cn/htmldom/prop_style_maxheight.asp) | 设置元素的最大高度 | 5M | 1 | 9 | Yes |
| [maxWidth](http://www.w3school.com.cn/htmldom/prop_style_maxwidth.asp) | 设置元素的最大宽度 | 5M | 1 | 9 | Yes |
| [minHeight](http://www.w3school.com.cn/htmldom/prop_style_minheight.asp) | 设置元素的最小高度 | 5M | 1 | 9 | Yes |
| [minWidth](http://www.w3school.com.cn/htmldom/prop_style_minwidth.asp) | 设置元素的最小宽度 | 5M | 1 | 9 | Yes |
| [overflow](http://www.w3school.com.cn/htmldom/prop_style_overflow.asp) | 规定如何处理不适合元素盒的内容 | 4 | 1 | 9 | Yes |
| [verticalAlign](http://www.w3school.com.cn/htmldom/prop_style_verticalalign.asp) | 设置对元素中的内容进行垂直排列 | 4 | 1 | No | Yes |
| [visibility](http://www.w3school.com.cn/htmldom/prop_style_visibility.asp) | 设置元素是否可见 | 4 | 1 | 9 | Yes |
| [width](http://www.w3school.com.cn/htmldom/prop_style_width.asp) | 设置元素的宽度 | 4 | 1 | 9 | Yes |

# Positioning 属性

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **属性** | **描述** | **IE** | **F** | **O** | **W3C** |
| [bottom](http://www.w3school.com.cn/htmldom/prop_style_bottom.asp) | 设置元素的底边缘距离父元素底边缘的之上或之下的距离 | 5 | 1 | 9 | Yes |
| [left](http://www.w3school.com.cn/htmldom/prop_style_left.asp) | 置元素的左边缘距离父元素左边缘的左边或右边的距离 | 4 | 1 | 9 | Yes |
| [position](http://www.w3school.com.cn/htmldom/prop_style_position.asp) | 把元素放置在static, relative, absolute 或 fixed 的位置 | 4 | 1 | 9 | Yes |
| [right](http://www.w3school.com.cn/htmldom/prop_style_right.asp) | 置元素的右边缘距离父元素右边缘的左边或右边的距离 | 5 | 1 | 9 | Yes |
| [top](http://www.w3school.com.cn/htmldom/prop_style_top.asp) | 设置元素的顶边缘距离父元素顶边缘的之上或之下的距离 | 4 | 1 | 9 | Yes |
| [zIndex](http://www.w3school.com.cn/htmldom/prop_style_zindex.asp) | 设置元素的堆叠次序 | 4 | 1 | 9 | Yes |

# 2月3

经过实验，style部分的处理基本上流畅。

按照css的分类，将每一个分类单独一个类来定义style的属性，每一个属性都设置其标志位AffectRender，增加一个——cssName的属性，然后在frameworkElement中通过OnPropertyChanged，检查如果是AffectRender，就直接在dom上通过setProperty（dp.\_cssName, e.NewValue）,从而设置其属性。

但是一个问题，就是对于border来说，如果是简化编写的话，setProperty(“border”, “solid; border-wirdth:thin;”)的形式好像不行。

嗯，如果严格遵守css的简写规则，是可以的，但是如果混合的写法是不行的。

# 小成果

呵呵，总算是实验好了css部分，这样的话，基本上就可以确定技术路线了，目前的实现，还是非常简洁的；曹永猛同志，你真厉害！继续加油吧。

阶段成果

Transform ok

Text ok

Layout ok

Transtion ok

现在剩下来的是Animation部分了；

# 实现dom元素

一个实验：div中被其他元素分割开的文字，如何处理呢？

猜测一下：首先这些文本肯定是不会被丢失的。

但是如何进行管理呢？

可以分别设置两个集合，一个是文本的，一个是元素的。

嗯，有问题，顺序不好处理。

还是统一一个，但是，逻辑孩子不好处理，需要进行区分。

视觉孩子还好吧。

在Html Dom中，可以插入多个TextNode到元素中，这些TextNode应该是不同的东西了。

并且这个TextNode居然也可以AddEventListener了。

# 问题

在基于frameworkElement构成的体系中进行编程的话，此时如何遍历对象树呢？

添加文本的过程中，自动创建TextNode来包装这个Text。

如果是元素，就可以直接添加进来。

当然，按照WPF的实现，这个地方又需要模板来解决了。

<div>

Sasasa

<input type=”text” />

sasasa

</div>

设计DOMElement这么一个基类。从FrameworkElement继承而来。

其他的html元素都是其子类。

每一个元素都有一个集合，保存其孩子。这个地方，孩子即是逻辑孩子，又是VisualTree的节点。

这个地方是两个集合，一个是逻辑视图的，一个是visual的，有些时候这两个集合可能不一致。

当我们添加一个孩子时，如果这个元素dom元素，就直接添加到VisualTree中，如果不是就只能是文本，然后通过封装为Span（TextNode）元素，添加进来。

最终在构造DOM元素时，是通过当前对象对应的dom，来构造的。最终组成了dom元素树，是否应该允许用户操作这个dom元素树呢？嗯，这个问题可以等到再往前走一段再说。

当然还是通过Measure和Arrange来进行的。这个Arrange主要是展开Template，Arrange主要是为了创建dom元素树。

当然，form元素还是需要从control继承而来，按照TextBox的方式。

这个地方的TextBox、

把Range是出入、Progress当成输出，

Input、Button、Check、Radio等当成输入。

这些dom元素都有一个特点，就是接收

从代码角度看，dom中的子元素可以是任何元素。

但是从html元素的组织角度看，这个元素就不好是随便什么元素了。

# Gradient部分

实验算是通过了，但是都带有自己的私有前缀，呵呵，不知道什么时候能够去掉了；

现在居然还只是Editor’s Draft版本。

# SVG的处理

在WPF中，只是封装了几个SVG的特性而已，因此既然决定采用html5，那么svg就有必要进一步丰富了。于是进一步浏览一下SVG，看看如何应对就很有必要了。

可以从FrameworkElement上继承，实现SVG的形状，只是这个形状，不再采用WPF的格式，而是采用SVG的形式，而且最关键的是，其属性可以和业务模型进行binding了。

在这个转换过程中，尽可能地不损失SVG的特性，而不是之前的做法，是对SVG进行了简化。

# 2月4

Canvas部分，从FrameworkElement上继承，增加一个获取Context的方法，在编译时这个方法，就直接是canvas的了，接下来就是浏览器的了，实现这个地方，并没有增加负担；

SVG，需要逐一实现SVG的元素，当然这些元素都需要从FE上继承，嗯，这个部分会增加很多元素，这些元素的注意事项，只能是嵌套svg的元素，不能嵌套其他的元素。但是这些svg元素，可以被嵌套在其他html元素中。

并且svg也可以嵌套svg。

大概的元素

1. svg、line、rect、circle、polygon、path、g、text、a、img、tspan
2. 样式有关的：transform、animation、stroke、fill、textStyle、linestyle
3. Def：

# SVG

Svg是从dom上来的，或者说是生存在dom中，自然会遵守dom的规则，何况，dom元素都是从node继承而来，操作dom的方法自然可以用来操作svg元素。

为了从编程的角度来看，好处理，需要收窄，需要按照svg的规范设计接口就好了，在内部，可以转换成dom操作；

呵呵，可以把每一个对象中的dom元素放出来，特殊时候，用户就可以进行操作了，这样意味着dom接口，也需要移植到dsl中才好。

# Html

实现几个html元素看看

Article

Section

H1~H6

# XAML

对于一个元素来说，如果嵌套了子元素，首先看看父标签元素是否实现了IAddChild，如果实现了，就直接添加到父元素上。否则就要报错了。只是此时对于子元素来说，此时的提示范围就比较广泛了。

所有的html元素也都可以实现IAddChild，当定义了子元素时，就添加进来，如果添加的是字符串，就可以直接添加当前集合中，在渲染时，如果是字符串，就创建textNode。如果是html元素，就需要添加到逻辑视图上，同时添加到视觉树上。

这个类中，需要重载ArrangeOverride，在这个方法中，建立dom树；

在维持逻辑树的过程中，才能维持dataContext之类的信息了；其实如果这个地方不继承，在binding的时候，可以主动来寻找，只是如果元素树结构变化，就没有继承策略好了。

DomElement，其孩子是FrameworkElement类型，或者是UIElement类型吧。应该是除了UIElement就是字符串，不能是其他类型吧。或者其他类型，最终都会通过ToString来显示其内容而已。

在其内部维持一个集合，允许维护这个集合。困难是，运行时如果增删这个集合的内容的话，如何维护已经生成的dom元素呢？而hhtml是具有这个特性的。

可以建立一个map映射。呵呵，这样也还是比较复杂地。

可以在元素上增加一个属性，来记录dataItem，然后通过遍历这个属性来查找对应的dom元素，从而维护其一致性。

有了Map后，就可以方便静态编译了，也就是不再完全是js，然后通过js来生成了，而是可以建立dom，然后在dom上进行初始化了。

# Summer

可以让FrameworkElement继承自DOM Element，首先具备这类元素的特征；

然后把其属性管理起来。事件、方法体现出来；

然后在这个基础上，添加FrameworkElement的一些东西，例如：DP特性——并且这个EffectiveValues实现在元素本身上，通过一个专门的值对象来保存。

实现GetValue、和SetValue（）这两个方法即可——当然其他一些附加属性也要实现进来；

这是DO、接着是Visual、UIElement、FE、然后是各种元素。

和自己实现的控件（例如：ItemsControl、Calendar等等）；

当然还有一种Control的元素，来实现ControlTemplate之类的特性；

然后把RoutedEvent采用DOM本身的机制来实现。进而实现Command之类；

很多时候，可能没有使用使用Control就搞定了。

呵呵，这个想法太彻底了，让这个思路沉淀一会儿先。

Visual这个类基本上就没有什么了；只要保留VisualTree相关的就可以了；这个地方最终可能会依靠DOM机制来实现。

UIElement主要是一些事件之类的信息；

FrameworkElement：维持逻辑树、ApplyTemplate、计算模板有关的属性等等；关键的几个地方：逻辑树、模板、逻辑parent、Style、 Focus、属性继承；

Control

ContentControl

HeaderedControl

HtmlElement

Article、Section、Header、Nav、Footer等等；

这样的话，在客户端基本上就可以混合dom元素的方法、和Summer对象的方法进行操作了；

还是有可能实现静态页面的形式来处理的。

视觉树就可以省略了，完全采用dom元素本身的机制来处理。——涉及到TemplatedParent的问题。

# A

|  |  |  |
| --- | --- | --- |
| [href](http://www.w3school.com.cn/html5/att_a_href.asp) | *URL* | 链接的目标 URL。 |
| [hreflang](http://www.w3school.com.cn/html5/att_a_hreflang.asp) | *language\_code* | 规定目标 URL 的基准语言。仅在 href 属性存在时使用。 |
| [media](http://www.w3school.com.cn/html5/att_a_media.asp) | *media query* | 规定目标 URL 的媒介类型。  默认值：all。仅在 href 属性存在时使用。 |
| [rel](http://www.w3school.com.cn/html5/att_a_rel.asp) | * alternate * archives * author * bookmark * contact * external * first * help * icon * index * last * license * next * nofollow * noreferrer * pingback * prefetch * prev * search * stylesheet * sidebar * tag   up | 规定当前文档与目标 URL 之间的关系。  仅在 href 属性存在时使用。 |
| [target](http://www.w3school.com.cn/html5/att_a_target.asp) | * \_blank * \_parent * \_self   \_top | 在何处打开目标 URL。仅在 href 属性存在时使用。 |
| [type](http://www.w3school.com.cn/html5/att_a_type.asp) | *mime\_type* | 规定目标 URL 的 MIME 类型。仅在 href 属性存在时使用。  注：MIME = Multipurpose Internet Mail Extensions。 |

# Area

|  |  |  |
| --- | --- | --- |
| alt | *text* | 规定区域的替代文本。如果使用 href 属性，则该属性是必需的。 |
| coords | *coordinates* | 规定区域的坐标。 |
| href | *URL* | 规定区域的目标 URL。 |
| hreflang | *language\_code* | 规定目标 URL 的语言。 |
| media | *media query* | 规定目标 URL 是为何种媒介/设备优化的。默认：all。 |
| rel | * alternate * author * bookmark * external * help * license * next * nofollow * noreferrer * prefetch * prev * search * sidebar   tag | 规定当前文档与目标 URL 之间的关系。 |
| shape | * rect * rectangle * circ * circle * poly   polygon | 规定区域的形状。 |
| target | * \_blank * \_parent * \_self * \_top   *framename* | 规定在何处打开目标 URL。 |
| type | *mime\_type* | 规定目标 URL 的 MIME 类型。 |

# 一个问题

就是Html元素的文本如何被设置上去呢？

在元素上定义的文本——xml中定义的文本，可以通过AddText的方式添加到Html元素上。

一个难题是在XAML上，一个标签被定义了多个文本，此时解析是否有问题呢？

这个AddText或者AddChild在FrameworkElementFactory中如何操作呢？

通过把文本作为FrameworkElementFactory的形式来处理。如果FEF是文本，可以直接进行添加即可。

可以在FEF中增加一个方法——不行，还是只能通过把Text当成一个FEF的形式才行，因为这个地方涉及到顺序问题，例如：

<p>

Text1

<tag1> sasas </tag1>

Text2

...

</p>

所以就需要把Text封装成FEF来处理。

# 2月5

继续思考html元素中的文本在FEF中的处理。

确实WPF已经实现了这个处理。

在InstantiateTree中，判断，如果text不为空，检查父元素是否为IAddChild，如果是，就直接把FEF的text添加到了父元素上。如果没有文本，就进入创建控件的流程。

也就是说，此时不管FEF是否有type，都不会创建type对应的类型，而是直接添加text到父元素上。

呵呵，微软的家伙，应该是早就碰到了这个问题。到省却了自己的麻烦。

在XAML的解析过程中，对于标签之前的文本，如果是在FEF中，就创建FEF，添加到当前的FEF中，对于标签元素，则创建对应类型的FEF了。

呵呵，如果这样，说明微软的FEF其实是可以创建FCE的内容的了。只是这个FCE就没有子元素了而已。到了FCE中，成了断头了。

# 问题

在domElement中的，需要合并处理Text和控件，此时该如何处理呢？

需要一个元素，还是借鉴一下微软的处理，叫做run吧，将来这个run就是直接对应TextNode的，或者说，run会创建textnode，从而把自己添加到父元素中，这样的话，就可以统一了。

呵呵，甚至允许其他类型的内容，如果不是UIElement、也不是String，意味着可以查找模板来渲染自己的内容吧。但是，这个地方缺少一个ContentPresenter——呵呵，还是有困难的。

如果需要能够自动渲染对应的类型，此时意味着需要使用ContentControl的控件才行。这个控件就多了一个ContentPresenter了。

# 文本的binding

在dom元素上，如何把文本binging到业务模型上呢。

因为不是DP，所以也就没有办法binding了。

此时这个Run就可以出场了，这个Run中就是一个Text的DP，从而也就允许Binding了；

如果直接创建了包含Text的FEF的话，意味着是隐含创建了一个Run；

也就是说，run对应TextNode；

Run具有data属性，这个类型是否也是从FE上继承呢？

应该是要从FE上继承，而不是DOM上继承。

# DOM Nodes

The entire document is a document node

Every XML element is an element node

The text in the XML element are text nodes

Every attribute is an arrtibute node

Comments are comment nodes

# Text is always stored in Text Node

A common error in DOM is to expected an element node to cintain text.

However, the text of an element nodes is stored in a text node.

In this example:<year> 2005 </year>, the element node <year>, holds a text node with the value “2005”.

‘2005’ is not the value of <year> element.

# 总结

目前的思路应该是正确的，网页看起来比较autumn简洁，而且可以自如处理html的所有元素。

# TextBlock

这个类就不再需要了；

假如到了ContentPresenter中，可以用Span来代替了；

# Input

为了保持与Html的一致，把所有的input统一到一个元素中，通过type类型来进行区别，这样的话，password元素也就不在需要了。

|  |  |  |
| --- | --- | --- |
| **属性** | **值** | **描述** |
| [accept](http://www.w3school.com.cn/html5/att_input_accept.asp) | *list\_of\_mime\_types* | 规定可通过文件上传控件提交的文件类型。 （仅适用于 type="file"） |
| [alt](http://www.w3school.com.cn/html5/att_input_alt.asp) | *text* | 规定图像输入控件的替代文本。 （仅适用于 type="image"） |
| [autocomplete](http://www.w3school.com.cn/html5/att_input_autocomplete.asp) | * On off | 规定是否使用输入字段的自动完成功能。 |
| [autofocus](http://www.w3school.com.cn/html5/att_input_autofocus.asp) | autofocus | 规定输入字段在页面加载时是否获得焦点。（不适用于 type="hidden"） |
| [checked](http://www.w3school.com.cn/html5/att_input_checked.asp) | checked | 规定当页面加载时是否预先选择该 input 元素。（适用于 type="checkbox" 或 type="radio"） |
| [disabled](http://www.w3school.com.cn/html5/att_input_disabled.asp) | disabled | 规定当页面加载时是否禁用该 input 元素。（不适用于 type="hidden"） |
| [form](http://www.w3school.com.cn/html5/att_input_form.asp) | *formname* | 规定输入字段所属的一个或多个表单。 |
| [formaction](http://www.w3school.com.cn/html5/att_input_formaction.asp) | *URL* | 覆盖表单的 action 属性。 （适用于 type="submit" 和 type="image"） |
| [formenctype](http://www.w3school.com.cn/html5/att_input_formenctype.asp) | 见注释 | 覆盖表单的 enctype 属性。 （适用于 type="submit" 和 type="image"） |
| [formmethod](http://www.w3school.com.cn/html5/att_input_formmethod.asp) | * Get post | 覆盖表单的 method 属性。 （适用于 type="submit" 和 type="image"） |
| [formnovalidate](http://www.w3school.com.cn/html5/att_input_formnovalidate.asp) | formnovalidate | 覆盖表单的 novalidate 属性。 如果使用该属性，则提交表单时不进行验证。 |
| [formtarget](http://www.w3school.com.cn/html5/att_input_formtarget.asp) | * \_blank * \_self * \_parent * \_top   *framename* | 覆盖表单的 target 属性。 （适用于 type="submit" 和 type="image"） |
| [height](http://www.w3school.com.cn/html5/att_input_height.asp) | * *Pixels* | 定义 input 字段的高度。（适用于 type="image"） |
| [list](http://www.w3school.com.cn/html5/att_input_list.asp) | *datalist-id* | 引用包含输入字段的预定义选项的 datalist 。 |
| [max](http://www.w3school.com.cn/html5/att_input_max.asp) | * *Number date* | 规定输入字段的最大值。 请与 "min" 属性配合使用，来创建合法值的范围。 |
| [maxlength](http://www.w3school.com.cn/html5/att_input_maxlength.asp) | *number* | 规定文本字段中允许的最大字符数。 |
| [min](http://www.w3school.com.cn/html5/att_input_min.asp) | * *Number date* | 规定输入字段的最小值。  请与 "max" 属性配合使用，来创建合法值的范围。 |
| [multiple](http://www.w3school.com.cn/html5/att_input_multiple.asp) | multiple | 如果使用该属性，则允许一个以上的值。 |
| [name](http://www.w3school.com.cn/html5/att_input_name.asp) | *field\_name* | 规定 input 元素的名称。 name 属性用于在提交表单时搜集字段的值。 |
| [pattern](http://www.w3school.com.cn/html5/att_input_pattern.asp) | *regexp\_pattern* | 规定输入字段的值的模式或格式。 例如 pattern="[0-9]" 表示输入值必须是 0 与 9 之间的数字。 |
| [placeholder](http://www.w3school.com.cn/html5/att_input_placeholder.asp) | *text* | 规定帮助用户填写输入字段的提示。 |
| [readonly](http://www.w3school.com.cn/html5/att_input_readonly.asp) | readonly | 指示字段的值无法修改。 |
| [required](http://www.w3school.com.cn/html5/att_input_required.asp) | required | 指示输入字段的值是必需的。 |
| [size](http://www.w3school.com.cn/html5/att_input_size.asp) | *number\_of\_char* | 规定输入字段中的可见字符数。 |
| [src](http://www.w3school.com.cn/html5/att_input_src.asp) | *URL* | 规定图像的 URL。（适用于 type="image"） |
| [step](http://www.w3school.com.cn/html5/att_input_step.asp) | *number* | 规定输入字的的合法数字间隔。 |
| [type](http://www.w3school.com.cn/html5/att_input_type.asp) | * button * checkbox * date * datetime * datetime-local * email * file * hidden * image * month * number * password * radio * range * reset * submit * text * time * url   week | 规定 input 元素的类型。 |
| [value](http://www.w3school.com.cn/html5/att_input_value.asp) | *value* | 对于按钮：规定按钮上的文本  对于图像按钮：传递到脚本的字段的符号结果  对于复选框和单选按钮：定义 input 元素被点击时的结果。  对于隐藏、密码和文本字段：规定元素的默认值。  注释：不能与 type="file" 一同使用。  注释：对于 type="checkbox" 以及 type="radio"，是必需的。 |
| [width](http://www.w3school.com.cn/html5/att_input_width.asp) | * *pixels*   *%* | 定义 input 字段的宽度。（适用于 type="image"） |

# 事件

继续处理事件吧

1. 键盘导航
2. 焦点
3. 键盘事件
4. 鼠标事件
5. 元素上的事件；

先实现mouseOver部分吧，现在基本上保证了元素和控件之间的1：1，因此此时鼠标事件应该比较容易了。

# 事件的冒泡和捕捉

当一个事件发生后，w3c标准是，先进入捕捉，然后再冒泡。对一个事件的处理，采用哪一种方式，取决于你的侦听器是注册在捕捉阶段还是冒泡阶段。

默认是冒泡阶段的。

呵呵，最关键的是IE11终于支持这个做法了。

这样的话，WPF中的事件机制，可以看作是DOM事件机制的WPF实现而已。

在WPF中，花费很大力气来解决事件路由的问题，呵呵，再次学习DOM的事件机制，才明白，这个并不是什么创新，只是旧瓶装新酒吧了，这个旧瓶还是别人的。

WPF定义了事件策略：冒泡、隧道、直接；

在DOM中，则是每一个事件都经过这么三个阶段而已。

先测试一下几个浏览器的事件阶段问题；

W3C规范：

event.stopPropagation()

Event.stopImmediatePropagation()

**说明**

该方法将停止事件的传播，阻止它被分派到其他 Document 节点。在事件传播的任何阶段都可以调用它。注意，虽然该方法不能阻止同一个 Document 节点上的其他事件句柄被调用，但是它可以阻止把事件分派到其他节点。

看看，这个地方需要把DOM 的event和WPF的event进行融合。

呵呵，终于明白DOM的事件机制了。还需要记住有一个stopImmediatePropagation方法，可以阻止同一个节点上的其他侦听器接收事件。

改造WPF，把隧道事件和冒泡事件合并为一个，还是把冒泡事件和隧道事件当成DOM事件的不同阶段侦听而已。

# Command

再次学习WPF的Command吧。

<Window.CommandBindings>

<CommandBinding

Command="myapp:MyAppCommands.MyCommand"

Executed="cb\_Executed" />

</Window.CommandBindings>

呵呵，看出来吧，Command是可以binding的。

从WPF的实现开，对于可以click的控件，当发生了click后，检查自己是否binding了CommandBinding，如果存在就执行Command了。这个就是Command的机制。

当然执行时，检查当前焦点元素，有当前的焦点元素发出CanExecute的事件，看看是否可以执行，如果可以执行，就随后发出Execute的事件，在事件侦听中，执行Command的Execute了。

上述就是所谓的事件机制，看上去还真有点麻烦，如果依靠html的事件，有没有问题呢？应该也是没有问题的，需要自己定义事件，然后DispatchEvent即可，注意这个事件也是有三个阶段的吗？

自定义事件中有几个标志：是否Bubble、CancelBubble等，设置好；

Var event = new Event(“eventName”);

AddEventListener(eventName, sasas, true|false);

Target.DispatchEvent(event);

有了这个机制，就可以实现WPF在Command部分增加的PreviewExecute、PreviewCanExecute、CanExecure、Execure等几个事件，从而实现WPF的Command机制了。

在UIElemnt中，有一个CommandBindings的属性，这个属性是一个CommandBindingCollection类型，也就是说，一个UIElement允许有多重CommandBinding。

CommandBinding：command、CanExecute、Execute、parameter、target；

ICommandSource：Menu、Button等都可能成为CommandSource——命令源。其实就是让target发出针对一个Command的Execute和CanExecute事件；然后分别让元素树上注册的侦听器来处理，这个侦听器就是CommandBinding——命令关联。

简单来说需要如下的步骤：

1. 定义ICOmmandSource，在其中的Command属性上选择Command；
2. 在元素树节点上关联CommandBinding

就算是完成了。

但是一个问题是：在一个如下的场景中，一个文本框，一个剪切Command、一个剪切Menu。

当我们选择文本后，如何让剪切Menu呈现出来可用的状态的呢？在何时的时机来调用CanExecute事件，在tergate上的合适时机触发CanExecute事件，来刷新界面上的状态显示；呵呵，这个地方还是应该比较复杂的；

# 2月6

在XAML中，如果属性类型为集合，判断后，通过Add的形式添加，需要处理Collection、List、Array、DIctionary几种情况。在XAML中已经处理了这种情况了；

# 家庭聚会

舅舅请大家在老客栈吃饭，徐文思没有去；舅舅一家去厦门过了年，初四回来。

# MVVM

Model、 View、ViewModel的简称，核心在于ViewModel，在这个模式中，View和Model感觉都很傻，只有ViewModel最辛苦。一般是在ViewModle中包装一个Model，从而来实现属性变化等，因为Model是为业务逻辑准备的，往往不是为界面逻辑准备的，因此还需要进一步进行丰富，添加一些界面交互需要的特性。

在计算器场景中，算子1、算子2、运算符、结果，其中运算符是一个Command，binding到了Button上，具体的运算则是由Command的Execute来完成的，这个Execute方法位于ViewModel中；要求Command是RoutedCommand才好；

# RoutedCommand

与一般的Command的区别，命令源发出命令后，路由事件交给命令目标来通过发出路由事件来执行，Command是直接执行Command。

# 事件

为了好处理，可以分为两个阶段，第一个阶段是转发dom事件来形成WPF事件，第二个阶段是完全使用dom事件；

Dom中的事件有自己的规则，如果完全覆盖dom的规则，按照WPF的规则来实现，未必是一个正确的选择；

# TabIndex

-1，0、大于0或者不设置。

通过这个属性来解决focusable的问题吧；

# AccessKey

WPF的访问键如何实现呢?

此时需要利用事件捕捉，例如：在TextBox上，注册捕捉阶段侦听器，判断如果是按下了alt和快捷键的话，就导航到那个注册了快捷键的控件上。

# 键盘导航

可以判断如果是tab，就可以屏蔽浏览器的默认行为，也就是屏蔽其他事件侦听器。这样打击范围就不是很大了；

WPF中有一个显示键盘提示的功能，就是在按下了alt后，把控件上的访问键显示出来。这个特性在html上，不好实现。在WPF中专门绘制到一个层上，到了DOM上就有点难度了，可以如下实现，采取accessTExt的思路，如果按下了alt后，可以遍历所有的accessText，让AccessText应用一个特殊的样式，在这个样式中，让Access Key醒目显示出来。

嗯，这个特性即使在office中，也不是完美显示出来，只有菜单的快捷键显示出来了。

# ListBox的键盘导航

怎么ListBox的键盘导航忽然可以了呢？可能是没有采用float的缘故，因为在这个样式中，容器的尺寸可能为空；

怎么firefox的listBox没有设置应有的属性呢？

# ScrollViewer的问题

自己来做一个滚动条，这样就可以避免滚动事件太频繁不好控制的问题。

同时也可以自己来控制line、page之类的滚动了；

# Scroll

这个Html的Scroll问题，看样子需要自己测试一下才行，如果只是看看相关的资料，还是比较难掌握的。

一个实验：实验的描述，容器div、内容div，容器div的高度300px; 内容div的高度800px; 通过一个按钮来改变容器div的ScrollTop。观察到的现象

1. 改变容器div的scrollTop位置，确实是让不可见内容可见了；
2. 当到达内容的最大高度后，继续改变ScrollTop，没有效果了，此时的ScrollTop的值定格在一个固定的值上不再改变；
3. 容器内容的滚动与滚动条之间不是必然的关系，本质是容器的scrollTop的改变，如果style设置为auto的话，只要内容的高度的超过了容器的高度，容器就会出现滚动条，没有办法直接设置滚动条的位置，滚动条是自动与scrolltop同步的；
4. 滚动条的位置，水平和垂直滚动条分别是从left和top开始，彼此不会互相覆盖；
5. 出现滚动条后，内容区域的大小自然需要减掉滚动条占用的区域；

经过实验，可以得出结论，完全可以自己实现一个ScrollViewer，在这个viewer中，可以自己实现一个滚动条，来处理滚动事宜；

# Gridlayout的动态性

实验一下动态添加和删除单元格的问题；如果这个行得通的话，意味着自定义工具条、和ScrollViewer就是可行的了。有了这个利器，意味着距离WPF的特性又近了一步。

先猜测一下GridLayout的动态性，如果单纯地增加row和column，应该不会增加表格。还因该对应地增加对应单元格的内容，如果对应单元格的内容存在，单纯修改row和column的数量，应该不会较少单元格。

花几天时间看看这个Scroll相关的东西吧；应该没有想象中的那么可怕了。

# ScrollViewer

需要具有按照物理像素和逻辑像素滚动的特性。

几个难题：如何初始化滚动条，由于这个滚动条依赖于其他dom元素的布局后尺寸，因此其设置就有点难度了，在什么时候设置了，能够正确设置吗？

逻辑滚动，交给panel，找到当前元素，找到下一个item，计算item的尺寸，然后设置容器的scrolltop为这个item的f（offsetTop）即可。需要设置已经滚动的item的数量和total之间的比例。

如果是按照像素滚动的话，其实可以直接使用即可。

# 2月7

一个问题：

对于Item类的控件，或者说，由ControlTemplate构建的控件，控件本身就不应该再有自己的dom元素了，否则与WPF的设计相违背了。

出现了问题：例如，当设置一个List的ItemPanel为Canvas时，此时需要在ListItem上设置top了left属性，于是问题出现了，这些属性本意是要作用到Item上，但是由于item本身创建了一个Div元素，于是就不对了，item的div上没有设置正确的left、top，于是canvas也就不能正确地处理了。

如果是不创建dom，会产生什么问题呢？

在添加孩子的dom时，会出现没有dom的情况，需要把模板展开的添加进来才行。

不过在孩子arrange后，肯定会展开模板了。

如果是创建了Button后，里面什么内容也没有的话，此时就没有必要展开模板了，这种情况下，dom就没有。

不过这个情况WPF也会碰到，可以看看WPF是如何处理的呢？

推测一下：肯定有一个ContentPresenter，只是没有进一步的内容了而已。但是一定是有dom元素了。

对于Item类的呢？也是同样的道理，肯定有一个ContentPresenter，因此也是可以的。

# WPF的模板

在ContentControl中，至少需要一个ContentPresenter，即使内容直接是UIElement，也是如此，需要通过ContentPresenter来包装一下才行。

在这个地方，微软的专家们已经纠结过了。定义在控件上的属性，最终需要作用在ContentPresenter上，不是还有个TemplateBinding吗。

呵呵，这个地方使用TemplateBinding，只是Content相关的属性对于ContentPresenter进行了自动binding，对于其他属性并没有；

在WPF中，定义在控件中的属性是可以参与布局的，因为Control虽然是空的，但是在Measure时，还是会使用定义在其上的属性。这个地方可以借鉴，但是需要把这些属性作用到内容dom上才行；

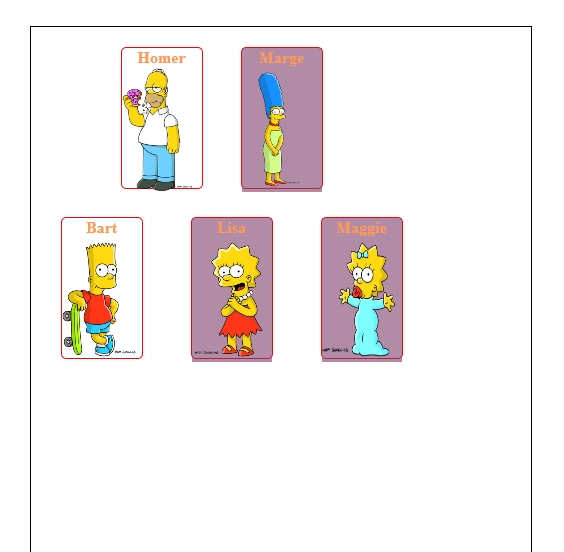
或者说，WPF中，外层容器确实不是透明的，还是在起作用的。

呵呵，忽然冒出来的这个想法是行不通的。

# 小成果

终于调试通了当ListBox的Panel为Canvas时的场景。在这个场景中，left、top坐标是通过binding来实现的。

截一张图留念：



实现的特性：

DataTrigger：gender为Femal时，背景色为分红显示；

Trigger：选中时，为蓝色背景；

顺便测试了子属性：Location.X和Location.Y的路径。

# WPF的简化

把Button从ContentControl中移走，可以独立使用ContentControl；

对于Group可以通过FieldSet来进行。而不再需要什么HeaderedContentControl的处理了。

那就改造一下看看吧；呵呵，如果让控件本身不再有dom，呵呵，试过了，这条路行不通。

# 开始DSL编辑器

先把Summer备份一下。

工作空间在C:\Users\永猛\workspace

采用Xtext2.5来编写DSL吧；

使用Wtp来编辑；

# 开始了

几个难点：

1. 嵌套的标记扩展表达式；
2. 嵌套表达式中的属性值。无引号字符串，其特征是从=开始，到，结束，可以利用这个特征吗？如果是=需要转义、如果是，需要转义；
3. 元素中间的文本数据；在一个标签结束，到标签开始之间的内容就是文本；
4. ContentAssistant提示机制是如何的；嗯，刚才跟踪了一下，即使是STRING也是随时可以进行ContentAssistant的。

既然是一个EMF模型，意味着可以采用EMF的编辑器来编辑一个XAML文件了；这个可以认为是办可视化的了；

# 2月8

DSL编辑器问

1. 为什么<Label />类的语法好像不能识别；

两个元素的结束和开始标签之间的文本的解析：

碰到结束标签后，记录下来，这是一个特殊的规则，中间的内容作为一个字符串，直到遇见开始标签为止。

在标记扩展中：开始与一个等于号，直到遇见一个逗号；中间需要转义处理，可以不去掉开始和结束的空格，但是需要去掉开始的回车换行才行。

就先从这个开始吧。

# Lexer

呵呵，一直感觉编译器很神秘，看了很多次，每次都是浅尝辄止，以致于积累了很多恐惧。这次，不经意间又遭遇了anltr，当看到其生成的解析代码时，一下子豁然开朗了

通过mTokens（）来获取下一个token，每一个token都对应一个lexer规则，于是parser就以此为基础登台表演了。

如果需要解析标签开始和标签结束的话，可能需要劳烦这个地方了。

nMtokens，调用了predict，来预测lexer规则。

Predict方法是一个循环，在循环中调用了specialStateTransition这个方法。

这个方法利用了生成的DFA，在这个DFA中预测lexer规则了。

# 任务

先实现一个XAML的简单编辑器，测试其他思路是否行得通。

# 感叹

人的记忆真的不靠谱，以前清清楚楚，明明白白的事情，现在感觉居然只是似曾相识而已。

# XAML

如何能够在标签中选择类型呢

如何让js的dsl和XAML能够共享呢

困难：java的方法签名包含了参数个数，——与js的机制不同。这个地方需要注意，在js的DSL中有多个方法针对不同的参数个数，但是在编译为js时，可以不管他。

# 又开始下雪了（11点40）

# XAML DSL

先不去处理PCDATA、以及Binding中的属性值问题。

# JsDSL——javascript的DSL

这个地方绕到了js的DSL，看样子需要先把这个dsl收拾一下才行。

让js具有类的特征；

需要属性、方法、可以支持泛型、需要集合的支持，但是所有的这些都是jsDSL来定义的。

需要事件的支持，类似于c#的手法；

属性部分可以类似于c#的语法，可以携带有属性变化通知的特性。

# JsDSL

先采用Xtend的语法，采用java的语法规则进行编译。也就是XAML变成一个JVMDeclareType，就可以让XAML使用xtend写好的那个类了；

让Xtend变成一个JvmDeclareType，就可以在XAML中进行参照了；

最终是通过IGenerator首先编译XAML的behind类编译为js类，然后编译XAML为一个html类，当然在html中需要引入behind的js类。

实现了编译后，在试图优化为更好的吧，例如：属性、事件等机制的语法糖；

# 问题

1. 解决如何在XAML中import需要的类。
2. 按照WPF的思路，首先把默认的包名称控件的类引入进来——当然在这个地方尽可能地保证名称不冲突，然后通过特定的ns前缀的属性来引入其他的包名称。
3. Js中所有的都是Object的子类，这个约定有无问题呢？应该没有，在xtend中这个问题可能不好处理。

# 分析

经过大半天的实验，虽然去年花了几个月时间在Xtext上，但是对其掌握仍有限，需要充分的思想准备，这又是一场硬仗。曹永猛同志，继续前进吧，为了那些期待眼神。

如果从XBAse上实现，很多基础设施好像都有了，但是XAML的语法与XBase的语法冲突太多，看样子是不可能将其建立在XBase上，因此只能白手起家才行。

好在其他的可以建立在XBAse上，呵呵，也不一定啊。如果能够熟练掌握Xbase的话，后面的很多障碍克服起来才有信心。

再次向XText发起冲击，要拿出研究WPF的劲头来。

为了不忘记WPF那一部分，可能定期要抽出时间回头看看。两者要交叉进行。

# XAML

如果不在希望在VS上编辑的话，为什么非要吊在XAML这个树上呢？

如果是其他形式不是也挺好吗？

例如：把《》换成def，

Def Page {

Aaa=a;

Bbb=b;

Def Page.aaa = {

}

}

倒也是一个办法。呵呵，主要是这个语法距离html比较方便，尤其是表达paragraph的内容时，如果用其他的手法，就有点不好理解了。

# Xtext

这个框架的内容感觉像一团乱麻，根本找到起点。

需要结合UI部分一起来理解才行。

# Guice

这个对东西也需要先复习一下。曹永猛，要动手啊。

这个Guice在RPC中可能用武之地。

首先来分析一下。

就是依赖注入，也就说当创建一个类实例时，能够把依赖的实例注入进来。

呵呵，首先不要神话，世界上没有任何的自动，如果没有实现准备好的脚本，因此guice应该也不例外。要想自动注入，首先需要定义接口和实现之间的关系，如果只有一个实现，应该是比较容易的。由于构造实现过程中，可能还涉及构造函数的参数，因此还需要一些配置。有些时候需要共享一个实例，有时要求使用不同的实例，这个也是需要之一。

是否可以不仅仅是针对接口呢？

呵呵，对于框架的使用，只有看一些著名框架使用guice的手法，才能更好的学习Guice的使用。

看到Xtext对Guice令人眼花缭乱的手法，真是佩服，总有人会把一个工具发挥到极致，甚至超出工具设计者的想象。

# Scope

这个概念非常重要，我们经常提到作用域，却很少在其上停顿一下思绪，想象这个概念意味着什么。就像我们生活中的那很多想当然。在XText中，Scope这个概念非常重要，可以说是理解xtext的关键，也是掌握xtext的关键。

在contentAssistant中、validate等地方都使用了这个概念。

并且处理作用域也是dsl实现的重要内容之一。

# Import

在dsl中总是需要使用其他文件定义的信息，此时就是这个import存在的意义了。

例如；在java中，我们需要使用其他类，呵呵，这个几乎是必然的，Object，此时就涉及到了如何让当前类知道Object类呢？此时就是import了。Import本质上是干了什么呢？其实是引入了一个名称空间，例如：import xxx.xxx.\*。可能是一下子引入很多个对象当当前的作用域中，也可能是引入一个对象到当前的作用域中。

Import tt.Page;

<Page Title=“” Background.Color = “rgb(1,1,1,1)”/>

此时这个Page就是当前作用域的一个对象，由于Page是参照的，因此如果当前作用域内如果存在，自然也就正确了。

Title认为是当前Page的一个Field，如果当前作用域中存在Title的field，自然也就正确了。这个Background是一个类型，因此需要被import进来，然后COlor是Background的一个Field，如果确实是这样，自然也就没有问题了。当到达Color时，由于已经存在了Background，因此，此时作用域对于Color来说，就是Background了。

# 2月9

其实Summer中的TextBlock不再需要了，只需要一个TextNode即可。这样，Summer又可以进行瘦身了，而这个瘦身可以说是根本性地。

# JsDSL

需要把js的Object、String、Number、Boolean、Date、Array等基本类型重新实现；

对数组的支持比较有限，把数组当作集合来处理了，当然jsDSL也可以这样，毕竟在js实现上，集合仅仅是对Array的简单封装而已，也并不影响运行的完备性——除了习惯。

呵呵，也就是说Array变成一个对象后，就是一个集合而已了，可以采用集合的方式来处理，而不再有js下的Array了。这个可是进一步简化了jsDsl了。

# XAML

这个进一步明确了，如果不在其中编写代码的话，这个dsl编译器应该比较容易。

并且采用JvmField后，属性的处理也比较简单，属性只能是附加属性、依赖属性、pojo的本身属性；

对于依赖属性一定是Property的、对于附加属性要通过setValue、GetValue处理，对于其他属性，就按照类本身的setter和getter来处理。

呵呵名称控件限定就不要了吧；

# XAML

想写一个最简单的编译器，居然在这个地方给难住了，写出来后，为什了import就是不起作用呢？

这他妈的奇怪了，花了大半天时间居然也没有找出原因。

看样子，还是静下心来，再仔细查找一下原因吧。

先比较看看Xtext的Example与自己的有什么不同。

# 大大的乌龙

折腾了一天，一直没有把自己写的XAML的dsl调试好，到了下午，才发现，原来是一个大乌龙。

在设计JvmTypeReference时，居然写作：[JvmTypeReference | QuantifiedName]的形式，其实JvmTypeReference 本身已经是[JvmType]的形式了。

# Scope

在XAML的属性部分，其Scope的规则是。

查询当前的上下文应该是Property，其容器应该是Element、然后试图获取Element的Type，获取到后，把这个类型的JvmMember作为上下文内容提供出来。

此时，属性应该不会出现找不到名称了吧。

嗯，又碰到一个问题。

就是属性的写法，在现在的写法中，[JvmType .] ? [ JvmField]

这个语法会有歧义，如果只有一个部分，认为是什么呢？是JvmType——没有输入后面的部分，还是JvmField呢？

最严重的是，可以在校验的时候，进行判断，如果只有一个部分，就当成JvmField，如果是两个部分，前面一个就当成JvmType好了。

如果是在Property输入过程中，只要还没有输入 period的符号，就认为是输入属性一，此时可以把类、和类的属性都列出来（当前作用域内的）。

一旦选择了后，

是在不行：就采取一个特殊的语法：的方式吧？用括号括起来怎么样呢？

# 2月10

今天是天天满23个月了。

争取在天天24个月的时候，summer能够在XAML中编写，然后编译为html端运行。

在XAML中，需要一个机制，区分属性元素、还是对象元素才行。

还需要区分依赖属性和附加属性的区别，在XAML的语法层面对这两个如果不进行区分，就不好处理。

如果附加属性也需要采用属性对象元素来处理呢？

<Page>

<Button />

<% Page.Content />

</Page.Content>

</Page>

约定好，采用（）依赖属性和属性元素要采用括号括起来，以便好解析吧；

只是稍微丑陋了一些。

嗯，还是先采用括号的方式吧。

既然已经破坏了XAML符合XML语法的规则，干脆就在打破一些XAML的规则吧。

对于标签扩展，可以采用{，不再添加放到引号内。

# 收获

开始练拳了，下雪让自己的太极拳中断了还几天。

今天早上，决定不再坚持XAML的规则的那一刻起，心头疼了几下，好像是不再完美了，习惯了一会儿，感觉就不错了，品味了一会儿，已经是心有得意了。呵呵，这就是多变的人心，飘忽不定的心情。

最新的语法：

**grammar** org.summer.markup.xaml.XAML **with** org.eclipse.xtext.xbase.Xbase

**generate** xAML "http://www.summer.org/markup/xaml/XAML"

**import** "http://www.eclipse.org/xtext/common/JavaVMTypes" **as** types

**import** "http://www.eclipse.org/xtext/xbase/Xbase" **as** xbase

**import** "http://www.eclipse.org/emf/2002/Ecore" **as** ecore

Document **returns** *Document* :

package = PackageDeclaration

;

PackageDeclaration:

'package' name=QualifiedName '{'

importSection=XImportSection?

(root=Element)

'}';

Content **hidden**(WS):

Element

|AttributeElement

;

Element:

"<" type=JvmTypeReference

properties+=PropertyBase \* => ((">"

(contents+=Content )\*

"</" close=JvmTypeReference ">") | "/>")

;

AttributeElement:

"<" "(" type=[*types::JvmGenericType* ] "." attribute=[*types::JvmField*] ")" =>((">"

(contents+=Content+ )\*

"</" close=[*types::JvmGenericType* ] "." closeAttribute=[*types::JvmField*] )">" | "/>")

;

PropertyBase:

Property

|AttachProperty

;

AttachProperty **returns** *Property*:

"("type=[*types::JvmType*|ID] '.' property=[*types::JvmField*|QualifiedName] ")" "=" value=XPrimaryExpression

;

MarkupExtenson **returns** *xbase::XExpression* **hidden**(WS):

{*MarkupExtenson*}

"{"

type=JvmTypeReference

properties+=PropertyBase\*

"}"

;

Property **returns** *Property*:

property=[*types::JvmField*|QualifiedName] "=" value=XPrimaryExpression

;

XPrimaryExpression **returns** *xbase::XExpression*:

XStringLiteral

|

MarkupExtenson

;

XStringLiteral **returns** *xbase::XExpression*:

{*xbase::XStringLiteral*} value=STRING;

# 计划

先把这个XAML编辑器调试好，然后先用xtend写一些js的类库。先编译出来html和js脚本。

估计xtend不能直接用来作为jsDsl，这个地方可能也是需要花费很多精力来改造，尤其是语法检查上，毕竟与java之间还是有点差距的。

# Resource

估计在XAML中，可以独立定义Resource，然后在Application中引入进来。最终最为隐含的样式资源吧。

# XAML的模板定义

<StackPanel.Resources >  
 <DataTemplate x:Key="MyTemplate">  
 <Grid>  
 <Grid.ColumnDefinitions>  
 <ColumnDefinition Width="128" Name="GoodsName" />  
 <ColumnDefinition Width="32" Name="GoodsUnit"/>  
 <ColumnDefinition Width="64" Name="Price"/>  
 <ColumnDefinition Width="64" Name="Quantity"/>  
 <ColumnDefinition Width="96" Name="Cost"/>  
 </Grid.ColumnDefinitions>  
 <TextBlock Foreground="Red" Text="{Binding Path = GoodsName}" Grid.Column="0"/>  
 <TextBlock Text="{Binding Path = GoodsUnit}" Grid.Column="1"/>  
 <TextBlock Text="{Binding Path = Price}" Grid.Column="2" HorizontalAlignment ="Right" />  
 <TextBlock Text="{Binding Path = Quantity}" Grid.Column="3" HorizontalAlignment="Right" />  
 <TextBlock Text="{Binding Path = Cost}" Grid.Column="4" HorizontalAlignment="Right" />  
 </Grid>  
 </DataTemplate>  
 </StackPanel.Resources>

这段代码在编译时需要特殊处理，需要把这个模板翻译集合FrameworkElementFactory的形式才行。

从细节角度看，在FrameworkElementFactory中设置的必须是依赖属性或者附加属性才行。

这个里面的key如何处理呢？认为是一种特殊的类似于附加属性的东西，或者就简化为附加属性吧，这样才好处理。

Template是从DispatchObject上继承的，不是DependencyObject，因此Template本身是没有什么属性可以设置的。所以这个地方就是特殊的了，其实也不是特殊的。

并没有说属性元素定义的对象就只能赋值到DP属性上啊。

在XAML中，只是要求在模板中，如果设置属性的话，就只能是依赖属性（或者附加属性）了。

在展开模板时，创建的DO元素，最后都是通过AddLigicalChild的形式来插入到逻辑树上，此时就交给具体的控件来理解这个AddLogicalChild了。

呵呵，在ColumnDeficiton中，有Name属性，这个Name属性可以用来干什么呢？

# XAML

Xaml的根元素是否要设置名称呢？

不需要吧，因为整个XAML的定义，其实就是一段创建对象的代码而已。

可以编译为一个大函数，在编译过程中，XAML中定义的命名控件，而html中需要有一个类，这个类来包装一个页面的含义。页面就是通过一个类实例，然后执行这个类实例即可，这个大方法也叫做InitializeComponent吧。在这个类中，在其原型上定义了XAML中的名称控件作为本身的属性。

# XAML——这个有问题(用前面的一个)

**~~grammar~~** ~~org.summer.markup.xaml.XAML~~ **~~with~~** ~~org.eclipse.xtext.xbase.Xbase~~

**~~generate~~** ~~xAML "http://www.summer.org/markup/xaml/XAML"~~

**~~import~~** ~~"http://www.eclipse.org/xtext/common/JavaVMTypes"~~ **~~as~~** ~~types~~

**~~import~~** ~~"http://www.eclipse.org/xtext/xbase/Xbase"~~ **~~as~~** ~~xbase~~

**~~import~~** ~~"http://www.eclipse.org/emf/2002/Ecore"~~ **~~as~~** ~~ecore~~

~~Document~~ **~~returns~~***~~Document~~* ~~:~~

~~package = PackageDeclaration~~

~~;~~

~~PackageDeclaration:~~

~~'package' name=QualifiedName~~

~~importSection=XImportSection?~~

~~(root=RootElement)~~

~~;~~

~~RootElement:~~

~~"<" type=JvmTypeReference~~

~~properties+=AbstractProperty \* => ((">"~~

~~(contents+=Content)\*~~

~~"</" close=JvmTypeReference ">") | "/>")~~

~~;~~

~~Content:~~

~~"<" (type=JvmTypeReference | ("(" attachPropertyType=[~~*~~types::JvmType~~*~~|ID] "." attribute=[~~*~~types::JvmField~~*~~|ID] ")" )) => ((">"~~

~~(contents+=Content)\* =>~~

~~"</"~~

~~elementTypeEnd=JvmTypeReference | ("(" attrElementTypeEnd=[~~*~~types::JvmField~~*~~|ID] "." attributeEnd=[~~*~~types::JvmField~~*~~|ID] ")" )~~

~~">") | "/>")~~

~~;~~

~~AbstractProperty:~~

~~Property~~

~~|AttachProperty~~

~~;~~

~~Property~~ **~~returns~~***~~Property~~*~~:~~

~~property=[~~*~~types::JvmField~~*~~|ID] "=" value=XPrimaryExpression~~

~~;~~

~~AttachProperty~~ **~~returns~~***~~Property~~*~~:~~

~~"("type=[~~*~~types::JvmType~~*~~|ID] '.' attribute=[~~*~~types::JvmField~~*~~|ID] ")" "=" value=XPrimaryExpression~~

~~;~~

~~XPrimaryExpression~~ **~~returns~~***~~xbase::XExpression~~*~~:~~

~~XStringLiteral~~

~~|MarkupExtenson~~

~~;~~

~~XStringLiteral~~ **~~returns~~***~~xbase::XExpression~~*~~:~~

~~{~~*~~xbase::XStringLiteral~~*~~} value=STRING~~

~~;~~

~~MarkupExtenson~~ **~~returns~~***~~xbase::XExpression~~*~~:~~

~~{~~*~~MarkupExtenson~~*~~}~~

~~"{"~~

~~type=JvmTypeReference~~

~~properties+=AbstractProperty\*~~

~~"}"~~

~~;~~

# XAML

语法基本上确定，是java和XML的混合体；

现在解决PCDATA的问题，可以采用Run元素过渡，等到需要锦上添花的时候，在完善吧。

注释采用java的语法

根元素上可以定义class属性，这个属性是附加属性。也可以采用默认的。如果采用默认的，就意味着与XAML文件名一致了。

这个类文件也是同步创建的，可以采用一个特殊的方式，来设置这个文件名称。

当然可以在XAML中链接去到这个文件。

<x:Array Type="sys:String" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:sys="clr-namespace:System;assembly=mscorlib">

<sys:String>Hello</sys:String>

<sys:String>World</sys:String>

</x:Array>

呵呵，又碰到了问题，这个String又需要Pcdata，而此时却没有办法是用Run元素了，呵呵，我引入一个Text的元素，来代表这个字符串。

嗯，现在应该比较容易解决了

可以为字符串再搞一个特殊符号，例如：#记号，嗯，先这样吧。可以是除了标签之外的任意字符。

郁闷，在XAML中元素Resources居然不是创建出来的，而是设置内容，呵呵，这个应该是普遍规则，集合属性一定是先创建好了集合，而是在XAML中定义其内容，如果是单一属性，则是创建一个元素来设置到这个属性上。

# XAML编译

找到根元素，是一个ObjectElement，然后找出其属性，以此创建这些元素，然后分别设置到外面元素的属性上。

依赖属性：可以翻译为

# IJvmModelAssociator

The Xbase framework will automatically switch between the JVM element or the DSL element when needed, e.g. when following hyperlinks. The component allowing to navigate between the source model and the JVM model is called IJvmModelAssociations, the read-only antagonist of the IJvmModelAssociator.

建立导航关联，用来实现超链，在Jvm Model和Source Model之间建立关联。

# IBatchScopeProvider

定义了这个接口，好像JvmField总算是可以了。

让我们来看看，这个IBatchScopeProvider到底是干什么的呢？

# 乐乐小朋友今天来了

11点来，3点走，跟天天玩的还可以；

# 2月11

XAML编辑器单独生成是有问题的，需要结合behind代码，今天看看Benhind代码的设计吧。

在Inffer中，只是推导Benhind的，首先在创建XAML编辑器时，同步创建这个Behind，在这个Behind中写入一个关联信息，也就是把XAML的文档关联到Behind。

在Benhind的构造函数中，执行XAML代码（——initializeComponent方法）。

用户的其他代码，可以用事件的形式来编写——或者Command。

需要有一个方法来模拟onload的。

InitializeComponent应该已经arrange完毕了才行。

因此可以执行设置初始焦点的问题。

同时生成Behind大概的代码，类名称、构造方法，InitializeComponent方法。还有XAML中的命名对象变量；

如果自己定义属性的话（或者修改已经生成的属性时）需要给出提示——应该是不允许修改吧。通过添加一个特殊的标注？

如果自己定义属性的话，要保证不能与XAML中的重名；

这个应该类似于WPF的做法，只是在内存模型中，并不在真正的behind代码中，但是InitializeComponent呢？也是在内存中。等到生成js代码时，才会生成有形的存在。

这个jsDSL才是困难所在，不过可以先用xtend来模拟，还可以在xtend的基础上来实现。

这写在内存中的对象，以Behind的某个属性形式存在，需要在Scope等地方照顾到这个存在。

# XsDSL

需要一种特殊的标注，表示方法是native的，也就是说是原生的，不需要实现。类似于GWT的手法。

需要在xtend中增加一个native的关键字，有了这个关键字的话，还可以允许在特殊的情况下，直接编写js代码了；当然这个代码需要被设置到一个特殊的标志下，类似于GWT的形式：

public static native int badExample() /\*-{

return "Not A Number";

}-\*/;

public void onClick () {

try {

int myValue = badExample();

GWT.log("Got value " + myValue, null);

} catch (Exception e) {

GWT.log("JSNI method badExample() threw an exception:", e);

}

}

# 事件

增加两个关键字，event、property，然后在语法中采用c#的语法。这样除了在XAML中设置一个事件外，还可以在代码中支持event的运算符形式来处设置事件了。

如果引入事件，意味着还需要函数对象是吗。还需要引入delegate关键字了。

Delegate负责定义一个代理对象，这个代理对象具备函数特质，可以在需要的地方直接以函数的形式来执行。

# Xtext限制

一定需要一个前导的字符，不能直接从id开始，或者不能有多个规则从id开始，这个限制太蛮横了。

可能需要花些时间看看，xtext的家伙们为什么突破不了这个限制。

以致于xtend的构造函数都没有办法按照java的语法来实现。

# Appendable

再一次学习这个对象，距离上一次学习差不多一年时间了。

这个对象几个作用，一个是把代码分成片段，方便中间插入；另一个管理临时变量；跟踪源代码与生成代码；

# 计划

先完成创建XAML时，同时创建一个xtend文件。

打开编辑器的时候创建的这个文件，先找到这个位置。

嗯，有问题，如果在创建XAML文件的地方来设置，意味着如果不是通过特定命令的话，就不能执行这个创建了，还是在XAML的保存中，来处理，如果已经存在，就不需要创建了，否则就需要创建一个，也就说在builder部分来实现的。

也可以在向导部分来完成。当然还需要对其他渠道创建的进行拦截处理。

在XAML中默认倒入这个于自己的名字相同的名称空间，并在内存中建立对这个js文件的引用，用户可以通过class来指定一个。

指定了这个文件后，获取到其内容的JvmModel，然后设置到当前的js中，这个文件的扩展名还是特殊一点好。并且把XAML的对象设置到js，先通过属性的形式来设置一个吧。在Page对象上，就有一个这样的属性先，是类型的。

这样就是显式处理了。

在选择这个后，同时在js上设置这个xaml对象。

这样就建立了双向关联。

先不用在内存中处理，这样的话，就可以显式处理了；

这个地方还是很复杂的，采用粗暴一点的处理手法，就是手工在两个文件中建立联系；

或者按照简单规则进行处理：

Xxx.xaml.js的约定，如果文件被删除，自然会被发现。

还是需要一个特殊的关键字：partial，检查如果是存在这个关键字，就自动从自己的文件名把xaml加进来。

默认是在xaml中寻找自己文件名对应的js文件，也就是固定存在一个这样的名称到导入；如此这般的话，自然可以手工建立了。

在js中获取本地scope时，计入xaml的内容。XAML的内容具有it和this的级别。

这样的话，就不要创建两个文件之类的复杂机制了。

但是引入的新问题：

是否可以通过默认的导入的名称空间，来使用XAML的内容呢？也就是是否可以建立一个内存中的crossreference呢？

# 设想

不要建立CrossReference，而是在创建Scope的地方再解决。

但是需要设置一个标志；

在ScopeProvider中，计算出来XAML中的变量定义，当前的Scope上。

在InfferModel中，设置上去；生下来的就是在Generator中生成需要的代码了；

类路径上的、其他插件里面的类定义——这个地方需要遵循OSGI的模块化机制了；这个没有问题，但是可能需要自己来处理所谓的classPath了。

呵呵，想多了，在这个地方，已经不再遵循OSGI了，因为最终会编译为js运行在浏览器。所以这个地方不能涉及jar，但是基础类在什么地方呢？可以在一个单独的文件内，例如：src/lib内；这个文件夹内的文件会全部添加到全局名称空间上；

# xtext想爱你也太不容易了，deprecated的实在生硬，变换了想法也不给台阶

一个场景

假设一个xxx.xaml.js，在这个场景中，确实存在xxx.xaml，那么此时如何与xaml交互呢，也就是说，需要添加到当前的ResourceSet中吗？

Xtext的scope有太多的类，到底这些类之间是什么关系呢？

也就是第一个入口是什么呢？

# 几个使用Scope的地方

Validator：

Linking

AssistantContent

QuickFix

# Linking

这个地方如何使用Scope呢？在一个上下文中，Scope会包含一些对象，此时这个Link就是与这些对象中名字匹配的对象建立link吧，建立的这个链接应该是暂时的，这个链接会在Hyperlink中使用，用来在引用之间进行导航；

在保存时，应该也会执行检查，查看是否CrossReference是否都有效；

建立连接时：应该先创建一个proxy，等到真正需要这个对象时，才会去寻找真正的对象。

platform:/resource/tt/src/tt/test.xaml#xtextLink\_::0.0.1.0.0::2::/2)

# Xtext

EMF的Resource和语法之间到底是什么关系呢？

或者说，在什么地方，一个文本文件变成了一个EMF的对象呢？应该是在解析过程中完成的吧；碰到一个解析规则，肯定是对应一个EMF的类型，或者说一个规则就是一个EMF对象来对应。那就先看看解析过程吧。

IParseResult，文本文件解析完成后，有两个对象，一个是对应语法定义中EMF类型对象的根元素，另一个是语法树元素。

也就是说，文本解析完成后，就有了EMF对象个抽象语法树对象。

在什么地方执行解析的呢？

线索：

在SetDocument中，创建XtextResource，然后把文本传进来，找到解析器从而真正产生文档的内容——此时就是EMF对象了。

有了XtextResource后，接下来干什么呢？

或者说link的时机是什么呢？

为什么link中做的事情，就是把非comtainment的Reference都设置为proxy，这个proxy记录了自己的位置。这个位置是包含了feature的索引，包含了集合的索引等信息。

为什么要这样处理呢？

通过XtextResource装在后，就开始UpdateInternalState()，进而LinkModel——》doLinkModel（）；

与Validator是如何结合的呢？

# Adapter

让一个对象可以具有一个原来没有设定的特性。

例如：可以让文件适配一个接口，从而可以显示在树上。

# EMF

这个EMF真是讨厌，用了那么多年，居然还是没有完全吃透。先攻克这个Adapter吧。

# 02-12

ResourceSet在xtext中是什么作用呢？

在创建XtextResource时，需要XresourceSet，其他的资源也需要加入这个ResourceSet吗？

# 一篇关于XText和GMF集成文章

从中获取的信息：

1. LocalScope：Resource内部的；
2. GlobalScope：Resource外部的；ResourceDescription

内部的对象的IObjectDescription在内容变化就刷新，外部的变化在Builder中维护；这个Builder可以维护增量和全量两种变化。

嗯，link到底干了什么呢？表面上看仅仅是创建了一个urifragment而已。

# 呵呵，我好像明白了

为Reference建立proxy对象，在真正需要这个对象时，会从Resource中通过getObject（）获取，在这个地方来解决proxy问题。这个地方是重载了Resource的getObject方法，是从Scope中获取对应的对象，至此，终于把Scope、Link等概念联系到一起了。

呵呵，花了这么多时间，真是有意思。

platform:/resource/tt/src/tt/test.xaml#xtextLink\_::0.0.1.0.0::2::/2)

这个就是一个编码的xtextLink，虽然可以通过这个编码，在当前资源中，尽快获取到当前的context上下文，和RFerence，从而来调用Scope获取到对应的对象了。

# 全局Scope

设想一下import，对于import，scope如何来理解呢？或者说要如何使用import的信息呢？

对于一个Resource，查找importURI属性的值，认为这个属性的值就是需要导入的。

然后对属性值形成URI，然后测试通过ResourceSet来测试URI是否有效，从而来确认倒入的信息；看样子到了xtend中，如果是static导入的，需要进行特殊处理才行？

大概的流程，对自己的Resource，找出其中的importURI属性，测试是否是有效的Resource的URI，如果是，就管理起来，使用的时候，通过本身的Resource的ResourceSet，来加来这些Resource，然后形成参照。

# XAML编译

如果是Page，就需要编译成Html了，这样不严格，应该是如果名字包好了xaml的文件，就意味着需要编译为html，否则就编译为js，或者都是编译为js，如果有xaml文件的话，就附加一个html文件的生成。

嗯，后面的这句就正确了。复述一遍：编译js时，判断是否存是xaml的behind，如果是，就附加生成html，否则就生成js而已。

# jsDSL

Js的语法：1 .toString()这个写法不行，包括true.tostring()；

可以怎么写呢

可以变成一个Number的对象，然后使用这个对象进行操作；包括boolean也是一个对象；当然Object是一个对象。

Var p=0；p=p+1;

undefined这个关键字需要吗，需要，好在可以通过null来处理，只是到了js中才会不一样而已。

Undefined是值不是对象。

但是允许函数重载的话，意味着在编译时还是有问题的，不是很好处理；

鼓励使用不同的方法签名，不再支持方法重载了；

为了编写程序时能够支持强类型，还是应该支持泛型比较好；

Xtend的有些东西就没有必要了；

# 天天会说虎、鹿、鸟鸟了

姨父姨妈、姐姐晚上回来了，有人举报不让补课。

# 2-13

方法不能重名、不能重载；

可以把undefined作为一个关键字引入进来；来精确处理，如果变量是否初始化了；

枚举需要改造，至少允许id、和value，从而可以设置一个指定的值，默认是采用一个从0开始的整数。

先把xtends的xtext自己生成一下，看看问题；

一个问题：

Java提供了放射来解决一些问题，jsdsl呢？需要一种方法来获得自己定义的属性吧；

嗯，可以使用Object上提供的方法。

嗯，还需要考虑到window是缺省的上下文的问题，也就是window默认是在当前作用域的祖先作用域上的；

可以获取一个对象的dom元素，可以操作这个元素——当然是按照js的规则了；

可以增加一个apply和call的方法吗？

不能给this赋值；

Object上的方法：create、defineProperty、DefineProperties等、以及一些内部属性\_\_proto\_\_等；

在jsdsl中，annonation可以不需要了吧；

泛型只是在编写过程中需要，到了编译为js后，这些泛型统统就没有了。

允许final出现，到了js中也是没有了；

常量定义：可以定义为property，没有setter、getter方法返回一个常数值；

对于私有的常数，是在类中通过var定义的形式来处理的，只是这个属性有了固定值而已，或者也是采用属性的方式；

=== ！==这两个运算符可以保留

构造函数需要特殊一点：不能有多个只能有一个，默认是会自动调用父类的构造函数，除非显式设置不调用super的构造函数；

可以通过标注的形式来处理，默认是调用的；

默认是是不调用super的构造函数，但是可以通过手工设置的方式来让其调用。

方法重载，就是会相互覆盖；

把def改为function关键字，这样的话更像js一些；

# 事件

单独为事件设置一个关键字？

事件处理就是一个类，事件中的属性类型也是类，当然这个类就是事件属性的类型了。

之后就可以采取普通的方式来处理事件了。

AddClickLsitener（Function<> Listener)

到了需要调用的地方，让这个集合是一个Delegate，将来统统编译为delegate相关的了。

可能还是需要一个特殊的关键字——event、或者标注来辅助编译；

如果是事件的话，就把Function这个类变成一个Delegate，从而来调用了。

在xbase中，是把Closure变成Function和Procedure，然后通过语法糖来进行；

到了js中是通过AddEventListner的形式来进行的，removeEventListener；里面是一个函数。

这个地方如何处理呢？如果是一个类，编译成方法有无问题呢？

创建一个侦听器，就是一个类，很多时候是一个匿名类，如果这个类有其他状态，可能就会有问题。

可以通过两个关键字，

一个是delegate、一个是event的形式来处理事件；

Delegate关键字其实是定义了一个特殊的类，保证了这个类不会有状态，并且可以直接作为方法被调用；

Property是针对属性的，event就是针对方法的特殊实现。

这个event只能是接着delegate，也相当于一个方法。如果进行调用，需要转换为某种形式才行。

# 正则表达式

Js中的正则表达式常量，看样子不好保留，可以采用regexp对象来处理，这样就可以避免regExp常量的语法问题了，问题应该会简单一些。

# Exception

在js中，并不需要在method定义中设置throws部分；

Try、catch、finally、throw

可以在方法上设置抛出的异常，只是到了编译时可以忽略这个声明，这样方便catch语句的检查。

在js中catch处没有类型，直接是一个变量，这个变量代表产生的错误。也就说，不能选择catch的error类型。

Throw可以抛出随便什么对象，当然也可以抛出异常对象了。

# Delegate

特殊的对象，可以当成方法来调用；

c#中；

Delegate returnType SayHello(type par1, type2 par2 ... );

event的类型只能是delegate的，delegate支持继承吗？不支持。每一个都是一个独立的类型。

可以对event的类型进行+-操作符来操作delegate的类型；

Event SayHello hello;

This.hello+= new SayHello(函数名字);

This.hello- = sayHello;

理论上来说，xtend是可以支持对delegate当成方法来进行调用的。或者对event进行直接调用的；

正常情况下可以：delegateType id的形式来定义的，然后可以new delegate（）的形式来初始化代理，或者进行设置的。

把他当成与enum一样的特殊类型吧。

# 闭包：

Javascript的闭包：setTimeOut（function（par）{}）；

也就说可以直接编译为function，而不是像xtend一样需要一个匿名类；当然在编译为java时，需要采用匿名类的形式，因为java中没有function这样的设施。

这个闭包，在JvmInffer中是体现不出来的，需要在compiler中，也就是生成代码的时候才需要体现；

在编译为java时是一个匿名的java类，到了js中，这个地方反而容易处理，可以直接

# 2月14

今天情人节与元宵节重合，算是百年一遇吧。

今天了解一下在js语法下的语法检查情况吧。

对于DP的定义；

Public static final XXX = aaa；

Property Prop{

Set{

},

Get{

}

}

属性的私有共有在js中是如何体现呢，体现不出来，只能体现是否可写；

有了属性后，就可以比较明确地实现属性变化了；

# 编译输出

面对一个dsl的类，需要固定的declare，其中的type等是固定的，然后是import需要的位置，然后是别名；

接着是构造函数、接着是其他的成员函数、然后是prototype属性定义；

然后是静态属性、静态方法、私有方法。

如果是静态方法，还需要在调用的地方进行对应的处理——添加类名称限定。

最后是type相关的处理，然后还需要看看是否有类初始化代码，如果存在，可以在一个一次性方法匿名方法中进行初始化。

如果是常量，分为私有和共有两类，如果是私有的话，就需要通过var来处理，如果是菲斯有的，就需要通过静态属性的方式来进行。

事件翻译为delegate的形式，也就是把add和remove替换+-；

让事件默认为undefined，在添加和删除时进行检查。

嗯，这样基本上就完整了。

用户接触的dom是另类的；

# Canvas

在这个地方是通过获取一个Context来进行的。

可以让Canvas对象能够代理Html的Canvas，从而释放出来API；

# Html的脚本片段

在html中，可以随时在html中嵌入一段js脚本，并且在这个位置，就执行这个脚本。

到了Summer中，应该不在具备这个能力了，除非允许在XAML中嵌入脚本，但是这个是很难的吧；理论上来说不可能，除非是要对应静态文本。

# Enum

在编译js时，enum的参照需要直接翻译为对应的值。因此枚举中允许设置字符串来代表其值。

# 一个问题

RPC部分

定义一个标注了RPC的jsdls的话，就认为只能是ajax调用了，这个jsdsl只能是一个接口，这个接口同时编译为java的接口，然后在java中实现，这种调用在编译时会生成ajax的桩代码，调用服务器上java实现，然后返回结果。

当然意味着中间涉及的bean都需要是两端都可以理解的。因此有一些bean，也需要采用jsdsl编写，但是此时对应java类型上需要找到一种机制，如何把js的简单类型变成java需要的精确类型。

还需要一种dsl，这个dsl可以横跨jsdsl和java。

从而完成二者之间的交互。

一种是业务模型、一种是rpc调用。

既然允许json，自然比较json更加强大一些就可以了。

还有一种语言，这类语言仅仅是为了输出js类的方法，不需要进行编译。

可以认为是idl定义，通过这个定义，各自的语言就可以使用这个定义了——接口定义语言；

一个bean，如果仅仅是接口的话，就有点问题，如果需要在jsdsl中创建的话，只是接口就不行了。看样子还需要一种稍微强大一些才好。

是否可以通过标注的形式在jsdsl上实现呢？有点麻烦，因为涉及到了java不支持的语言特性。

还是一种简单一些的比较好。

在java实现中，吧javabean变成json，传送到客户端，这个json需要被识别出来为jsdsl的类才行。

因此中间的这个转换处理比较关键。

但是理论上来说还是可以操作的。

所以要求这个bean尽可能的简单，除了方法就是属性，最多有静态方法和属性，需要采取java和js都支持的语法。

用这个语言编写好后，可以被java和jsdsl都能使用。

也就是这个语言需要编译为java、js两种目标语言，并且还需要能够自动处理二者基于json的转换。

这个语言语法应该基本上不允许表达式的存在。如果需要变化侦听，可以在java和js上各自封装一下即可。这样到了json中，也是没有问题的。

例如：调用rcp获取一个集合，然后在变换这个集合，添加变化侦听机制，在提交到rpc前，可以从包装实例中，获取原始的bean，然后把这个bean传递到服务器端。

如果涉及到代码翻译之类的呢？

# 一个场景

人员基本信息：

省市县的选择，在省列表框中，可以选择省，在省的选择后，影响到市，从而清空市，在市的变化中，清空县。

在省的列表框上，弹出popup，这个popup banding到一个集合，这个集合是的内容是动态变化的。当选择完成后，集合中要么有一项要么是空。而省输入框的值，binding到了省列表的popup控件的选择的项目上。

从而解决了autoComplete的问题。

也就是实现一个autoComplete控件。

当然这个控件是采用WPF的思路装配出来的。

是单向binding

增加孩子应该没有问题了！

照片应该可以；

性别选择器，也是没有问题的；

单选按钮的形式也是可以的；

焦点转换：没有问题；

验证：也是没有问题；

输入身份证、姓名调入已经录入的信息，没有问题。

提示保存成功：ok

年龄等几个字段的关联约束检查，没有问题；

一个列表，选择编辑、显示详细信息：没有问题；

输入多个照片：没有问题；

# Def

这个关键字还是改为function更好一些。

# 静态页面

2010年有专家预测，html5到2022年才会成为w3c的推荐标准，可是到了

# 再捋一遍

解析后，通过link创建proxy给Reference参照的对象，然后到了需要使用这个reference的时候，就解析这个proxy了。寻找到resourceset，获取fragment的第一部分，获取对应的Resource，然后找到上下文，默认通过ILinkService来链接Reference，这个ILinkService的默认实现是通过IBatchScopeProvider接口的一个来获取的。

在解析过程中，已经自动把import进来的名称控件添加进来了。

这个个名称空间认为是对应到resource的，需要通过ResourceSet打开，并获取其中的IObjectDescription了。

# 2月15

Scope的表达：

IScope代表一个具体的作用域，里面包含了当前作用域提供的元素；作用域通过嵌套组成上下一个上下文，里面作用域的重名元素会覆盖外部作用域 的。

IScopeProvider：作用域提供者，为一个具体的上下文提供作用域。

作用域从resource的边界角度分为三种：

1. LocalScope：当前上下文的作用域，最内层的作用域；
2. Import引入的作用域：通过import导入的作用域；
3. 全局作用域：默认、类路径、工程配置等引入的作用域；

在解析dsl文本时，碰到importURI属性，就创建IResourceDescriptor来导入对应的资源代表的作用域。当然会同时写入ResourceSet的URIMap吧，当需要根据一个URI获取资源时，就会通过这个Map来查询。

一个资源的IResourceDescriptor，其输出的所有的name属性的对象，每一个对象通过其containment关系，与容器的name组成自己的NameSpace的fragment。这些对象从EMF的角度看，需要是被当前resource存在ContainerComponent关系的，并且其存在所谓的name属性（当然每一个Eobject可以通过IQualifiedNamespaceProvider来提供自己专属的namespace 的fragment的）。

此时这些个importURI的resource，就直接作为了ResourceSet的资源了。

现在的问题：

类路径相关的IResourceDescriptor是什么机制?也就是说，Xtext是如何利用java的。从JDT的角度看，java元素可以以对象的形式获取，但是还是需要被转换为Resource的EObject才行吧，这个地方还没有看到。

猜测一下：

通过懒加载机制，可以把Workspace中的资源先建立起来，还需要在workspace中通过Builder来随时维护。这个可以看到，就是BuilderParttant了。但是没有看到是如何处理EMF的Resource的。

IJvmInffer还需要深入理解？与Scope是什么关系？

## IPackageFragmentRoot

JDT学习之IPackageFragmentRoot

IPackageFragmentRoot是一组IPackageFragment的根，代表源文件夹，也可以是jar包或zip包。  
比如Java工程中建立的src文件夹，某个依赖的jar文件 (比如rt.jar文件)。

# Xtext

呵呵，奋斗了几天，还是没有明白IContainer相关的处理。实在惭愧。

先在回头找找还有没有其他的线索呢？

一个Java类是否需要一个Resource来对应呢，也就是需要变成JvmType相关的类型。如果按照现在大概的理解，就是在每一个Project中建立对应的所谓的index，这些index最关键的是要处理好URI，然后就可以了，采取懒加载机制，等到真正需要使用的时候，在加载这个资源，其实也不复杂，因为在java工程中，已经建立了对应java类的模型，只需要把这些模型变成jvmType模型就好了。

但是关键是这个处理的代码会在什么地方呢？

嗯，一个线索是，这个地方可以通过注册Adapter的形式来从对应的IJavaElement元素上获得对应的JvmType对象。

这个IPackageFragmentRoot很关键，是一个入口，如果是file，就可以建立URI来对应了。

有了这个URI后，总是需要到ResourceSet中来检查使用这个URI了。

# Xtext

在CrossReference中，任何EMF对象，并不一定是JvmType。

不同的DSL，如果其使用相同的EMF类型，意味着就是可以进行CrossReference的，此时只要import另外一个进来，就可以CrossReference了。并不需要额外的工作。

在Xtend中通过HyperLink可以去到jar定义的class，因此说明二者之间肯定需要联系。

但是这个联系是怎么样的呢？为什么我就是看不到，金鱼，金鱼帮帮我，如果这个地方我搞清楚了，我一定写一篇长博客来讲述Xtext。

如果是自己的DSL，如果能够被其他的dsl参照，如果这个dsl是打开的，意味着并不需要做太多的工作？

但是有一个问题：对于dsl，需要保证使用何时的parser来解析，然后生成resource才行。

这个地方在在什么地方了？

打开文件后，解析这个文件；在打开Editor中打开文件时，猜测一下，应该是通过扩展名获取解析器，解析这个文件，由于都是XtextResource这个类型的资源。

# 还有一个线索

看看xtext上建立的工程是否有可借鉴的呢？

# 尼玛，TextEditor

我先花些时间动手解决一下这个拦路虎。

呵呵，这个东东就好像功夫你面的那个村妇，是卖大葱的，不好欺负。

# 呵呵

忽然发现在xtend中居然可以如下编写

1. toString();

也算是一个意外吧，这样的话，不涉及jsdSL编辑器都不行了。

# Xtext

啊，这个门槛是在太高了。曹永猛，咬咬牙，坚持一下，但是需要多动手才行。

学而不思则罔，死而不学则殆；

# Generator

Generator：是针对Resource的，因此不一定非要生成java代码，完全可以生成任何需要的代码。

# Synthetic

Synthetic人工的、合成的。

# IJvmModelAssociator

在DSL元素和JvmTypes元素之间建立关联，可以用于navigation等操作。

# Xtext

在Xtext这一层，还没有涉及到JvmType的概念，只有到了XBase中，才在表达式中贯彻了这个概念。

# 2月16

XtextResource

在这个类中，有一个Parser的setParser（）是一个@Inject类型的额，在RuntimeModule中注入当前dsl产生的解析器对应的Parser实现。

到了Editor中，在setDocument中，创建一个XtextResource，来对应当前的dsl文本文件，在加载这个文本文件的过程中，就完成了解析。故事也就是从这儿开始的。

解析完成后，使用Linker来对当前的CrossReference来进行link，其实质是为每一个CrollReference来创建一个Proxy——呵呵，正是这个proxy才成就了scope之类的神话。

Link后，resource就算是完成了，接下来是什么呢？

是否应该到了Editor中了，还是到了Builder中了。

在Xtext的service包中，有一个DefaultRuntimeModule，在这个模块中，配置了：

**public** Class<? **extends** XtextResource> bindXtextResource() {

**return** LazyLinkingResource.**class**;

}

也就是说，resource采用的是这个类型。

在LazyLinkingResource中，在load完成（parse、linking）后，首先访问一下CrossReference，来解决proxy的问题。

这样创建完成后，是否就会显示出来错误信息了？

# Xtext

通过跟踪调试，看到的情景：

Generator在Builder中，当发需要Builder时，其实就是执行Generator之时。

XTextBuilder实现了 IncreamentBuilder，注册为Xtext的builder类型。这个地方是通过IExtensionFactory来注册的。

在这个里面调用了XtextBuilderParticipant，从而执行xtext的builder逻辑。

注册Xtext的BuilderNature和Builder是通过org.eclipse.xtext.ui.shared插件来进行的。

这个地方需要进一步明确，builder维护索引，仅仅是针对变化的资源。

对于java中的类的参照，是在什么地方完成的呢?

猜测一下：

例如在xtend文件中，会加入了默认名称空间，这些资源应该是懒加载的，在解决CrossReference的过程中，Scope会加载这些Resource，从而完成的。

设置默认的导入的地方应该在xbase中，xtext不合适来处理这个问题。

有了uri后，在ResourceSet中加载uri对应的资源。

Import String类型后，就在当前的ResourceSet中建立了对应的URI映射，在需要的时候，会通过ResourceSet来打开对应的资源。

# 2月17

5点，窗外下着雨。

JsDSL需要自己从xtext上实现，意味着关于js的类型系统、语法检查、代码生成等都需要自己处理，呵呵，意味着自己要实现xbase和xtend，好在可以借鉴xbase、xtend，否则应该是一个不可能完成的任务。

测试了一下：

Xtype、xbase、xannonation这几个语法文件，都没有问题，顺利在xtext的基础上生成了代码。

# 静下心来

还是不能急躁，需要静下心来，虽然这很难，但是是唯一的正确方向。

# 儿子会逗爸爸了

儿子手里拿了一块糖，我表示出很想吃的样子，儿子就把糖塞到衣服里，张开手给我看，糖不见了，我显得很惊奇的样子，上下找，儿子就笑得咯咯的。

# Xtext

IScopeProvider：顾名思义，作用域提供者，直观的使用是在ResourceSet中，在获取CrossReference时，是通过Scope来获取的。

当然IScopeProvider是通过guice注入的。

最简单的实现是：就是默认注入了SimpleLocalScopeProvider这个实现。

这个类是从AbstractGlobalScopeDelegatingScopeProvider继承，其父类名字中有一个delegate，主要完成了把最外层的ScopeProvider的获取交给了IGlobalScopeProvider来处理，与当前的SimpleLocalScopeProvider组成父子关系的作用域。

IJvmModelInffer这个接口

Model inference is done after a resource has been loaded. It is used to map you language's constructs to the elements defined in [TypesPackage](http://download.eclipse.org/modeling/tmf/xtext/javadoc/2.3/org/eclipse/xtext/common/types/TypesPackage.html). Those elements closely reflect the Java type model.

For example if you have a language representing state machines with its states, transitions and actions, one way to map them to the types model would be to create a single class per state machine, an Enum for the different states, methods for the various actions and one method which contains the state machine logic. You would do that mapping in your language's implementation of this interface.

The inferred Jvm type model is used to compute how scoping is done and how the language compiles to Java. That is having a proper implementation of this interface is often the only thing you need to do to make your language executable on the JVM.

Implementers should use the [JvmTypesBuilder](http://download.eclipse.org/modeling/tmf/xtext/javadoc/2.3/org/eclipse/xtext/xbase/jvmmodel/JvmTypesBuilder.html) API to create a model of Jvm elements in a convenient way.

Implementors have to traverse the given source object in [infer(..)](http://download.eclipse.org/modeling/tmf/xtext/javadoc/2.3/org/eclipse/xtext/xbase/jvmmodel/IJvmModelInferrer.html#infer(org.eclipse.emf.ecore.EObject, org.eclipse.xtext.xbase.jvmmodel.IJvmDeclaredTypeAcceptor, boolean)) and decide which types to create according to the state of the model.

void **infer**(org.eclipse.emf.ecore.EObject e,

@NonNull

[IJvmDeclaredTypeAcceptor](http://download.eclipse.org/modeling/tmf/xtext/javadoc/2.3/org/eclipse/xtext/xbase/jvmmodel/IJvmDeclaredTypeAcceptor.html) acceptor,

boolean preIndexingPhase)

This method is called at two different times in a resource's life-cycle, reflected by whether {preIndexingPhase} is set to true or false. When set to true everything is still in a pre-indexing phase, that means linking hasn't been done yet. In this phase you just need to create the Jvm-elements which should be indexed (i.e. found globally). For regular Xbase expressions only the JvmTypes with the correct qualified name are needed at this point.

You must only infer Jvm elements which directly result from elements contained in the current resource!

When this method is called with preIndexingPhase set to false, you need to do the full inference including setting all links. But still you have to it in a particular order. First you need to create the complete tree structure and make sure you have passed the created JvmTypes to the acceptor. Only pass top level [JvmType](http://download.eclipse.org/modeling/tmf/xtext/javadoc/2.3/org/eclipse/xtext/common/types/JvmType.html)s to the acceptor. Only if the tree structure is created and its root types have been passed to the acceptor, you are free to resolve any cross references.

**Parameters:**

e - the root element from the parse result

acceptor - an acceptor awaiting derived root [JvmDeclaredType](http://download.eclipse.org/modeling/tmf/xtext/javadoc/2.3/org/eclipse/xtext/common/types/JvmDeclaredType.html)s

preIndexingPhase - whether the call is done in before indexing. During this phase clients may not rely on any global indexing information and only need to to provide the information needed by the language's[IDefaultResourceDescriptionStrategy](http://download.eclipse.org/modeling/tmf/xtext/javadoc/2.3/org/eclipse/xtext/resource/IDefaultResourceDescriptionStrategy.html). IF not implemented differently this is just the [JvmDeclaredType](http://download.eclipse.org/modeling/tmf/xtext/javadoc/2.3/org/eclipse/xtext/common/types/JvmDeclaredType.html)s with their qualified name, but no members and no other data.

# Xtext

一个DSL对象通过JvmModelInffer后，有两个，一个是当前dsl概念对象的元素，另一个是jvm元素的对象。

当我们导入一个名称空间后，其实两者都存在可能，xtext是如何协调这二者的呢？

这个地方应该是体现到xbase中吧。

Index是由IResourceDescriptions来维护的。这个要记住。

在这个地方的几个类：

IResourceDescription

IObjectDescription

这个index由IResourceDescriptions构造要坚持，然后试着攻克这个接口的相关实现，希望能够尽快突破。

有一个ISelectable的接口，IResourceDescription和IResourceDescriptions都是从这个接口继承，这个接口的核心是对外提供IObjectDescription的查找。

忽然想到的，在xtext层面，其Builder应该不会涉及到java类的索引建立。这个地方应该在xbase中或者xtend中才合适。在xtext中应该只是dsl本身的模型需要被索引。

IBuilderState继承自IResourceDescriptions

Jar中的class是在什么时候会被dsl使用呢？

在运行时，此时dsl已经编译成了java类，也就不需要ResourceDescription了，只有在UI编辑的情况下，才会需要。

跟踪调试发现，jre的类，确实是变成了JvmType表达的形式，从而为dsl使用，在什么地方JavaElement变成了JvmType的呢？

# XAML

只需要使用jsDSL的type即可，可以不使用JvmTypes。

# 呵呵

怎么就一直没有想到呢，把java类转变成JvmType相关模型的处理在org.eclipse.xtext.common.types相关的包里面呢。啪啪脑袋，不应该啊，既然是把java类转变成JvmType，从设计的角度看，自然应该在Common.types中才合理啊。

还是不够冷静，没有深入分析。

总算是找到了，在这个冬天里雨夹雪的下午，经过了好几天的痛苦，虽然相信只要坚持痛苦就会过去，只要时间够久，任何痛苦都会过去。

那就开始分析一下吧。

在JsDSL中，类型的这个机制不需要了。

由于是需要编译为js的原因，OSGI机制不需要、插件依赖不需要，并且class、jar也不需要。

Js需要的库都以dsl语言的形式存在。当然这个js也可以使用JvmType；

对于采用idl编写的接口，可以编译为JsDsl需要的形式。

由于接口中使用的业务模型，需要采取单独的语言来编写。接口采用一种特殊的语言。

业务模型的bean，需要以java和jsDsl的形式存在。

又绕到插件上了，如果在一个插件中编写了接口，或者业务模型，在另外一个插件中是可以使用的，将来这些插件参照的类都需要被添加编译进来即可。

如果在多个插件中的jsDSL，在编译时还是需要忽略插件机制的。

嗯，如果在jsDSL上实现插件机制，呵呵，不敢这样设想。

约定jsDSL只能在一个project中使用，不允许使用插件。换句话说，是jsDSL工程，可以存在工程依赖，但不是插件工程。

使用其他工程定义的jsDSL文件是可以的，但是其机制是工程依赖。

或者在创建js工程后就把一个lib文件设置到默认的导入中。

如果不能允许工程依赖的话，对于rpc的处理就不好办了。业务模型也不好办。

在创建好业务模型后，会自动生成jsdsl对应的类，这个类会单独编译出来js类，在这个类中，允许对json下序列化进行便利化处理。

理论上来说，是没有问题的。到了浏览器端的话，如何处理呢？可能需要自己的js库来处理才行——一个大任务。

因为涉及到java的精细类型，因此在json中，还需要插入类型才行。这个问题暂时先不考虑。

业务模型生成java类——没有问题。同时还需要能够有序列化、和反序列json的代码才行。

业务模型生成jsDSL的类——这些类通过工程依赖的方式来输出给需要的jsdsl工程，jsdsl工程中，就可以像使用自己的类一样来使用了。

也可以在jsdls中编写，然后导出到其他插件。也就是说，此时jsdsl工程是一个插件工程才行。并且要求是一个java插件工程才行。还要携带jsdsl工程的特性。

这样会简单一些。对于生成的java类，可以通过插件机制导出去了。

可以设计一种语言，其实是一种映射，一边是java的类型、一边是js的类型，然后在这两者之间建立映射。

# 打包

编译好后，打包、然后部署到服务器上，由于都是静态脚本（从服务器的角度看）和servler（需要通过某种方式配置），创建web.xml，在这个里面配置servlet，welcome等，基本上就可以部署了。

# Guice

终于找到了java的class是如何变成JvmType的对象的，心情高兴，回过头来深入搞一下guice。

1. binding多个实现到一个接口，Guice不允许，会出错；
2. Binding子类到父类，没有问题；可以覆盖了。
3. 自己不能binding自己；
4. 可以binding到自己创建出来的对象实例上；
5. 可以在方法上设置inject，例如：setXXX，会把参数需要的类型创建出来，注入进来；
6. 可以在构造函数上标注inject，会把参数对应的类型创建好实例注入进来；
7. 可以在字段上标注，会把字段对应的类型注入进来；
8. 可以注入静态字段；
9. 可以注入多个实现，通过Named Annonaton来区别；
10. 可以通过Provider来提供；
11. 可以注入常量；
12. 可以设置Scope；
13. 还可以binding Properties——注入常量的一种特殊形式；

总结了上面的知识点后，总算是一窥Guice的全貌了。

# 2月18

继续从Xtext上制作Jsdsl吧。

有了Guice的基础，来看看xtext的一个处理；

分别针对IScopeProvider和IBatchScopeProvider来进行了注入，而IbatchScopeProvider是扩展了IScopeProvider接口。这个场景下是什么情况，其实也很好理解，甚至可以推导出来。

并不会因为IBatchScopeProvider是从IScopeScope继承而来，对IBatchScopeProvider的注册会影响到IScopeProvider。

可以认为二者之间没有关联。

回到XBase的实现中，分别注入了这个接口的实现后，可以认为只有旧的代码才会使用IScopeProvider了，而新的代码应该使用IBatchScopeProvider了。

# XBase

在这个代码里，神秘莫测的就是Scope部分了，这部分代码混合了deprecated、新的思路的等等，让代码变得很艰涩。

这个地方可以通过FeatureCall为线索来揭开这团乱麻。

如果是要复制xbase的代码来实现jsdsl的话，会有些问题，尤其是需要同步xbase的升级。太多的代码需要处理。

可以只是先替换generator部分，来生成js看看。

# JvmModelAssociator

利用了Adater的方式，在Resource中扩展了一个Adater，在其中通过Map来保存source和target之间的对应关系。也就是说，在通过ModelInffer推导出来JvmType后，在模型所在的资源上建立起来关联。

嗯，JvmModelInffer机制还是比较复杂的，把推导出来的JvmType对象加入到模型所在的Resource，然后在二者之间建立关联。这个处理会有很多需要解决的问题。

IJvmModelInffer在IJvmModelAssociator中，可以推测出来，这个类是用来配合Generator的，因此在Generator中需要处理index等。

# Generator

这个地方的处理还是很复杂的。

# Guice

可以以Guice的处理为线索，换个角度看看Xtext工程的结构。

# Guice

需要从Injector获取对象的原因是使用Guice作为容器，如果不是通过Guice来获取对象的话，就和Guice没有关系了。

# Xtext

XtextDocumentProvider，中创建了XtextResource，加载、Linking、resolve。

XtextResource的加载过程，就是采用Parser来解析的过程，解析的过程中，创建EObject元素，这些元素按照语法规则组织为一个EObject对象，然后把解析完成的EObject加入xtextResource中，当然，需要清理之前的EObject对象——假如有的话。

IGenerator：在BuilderParticipant中使用的，当资源发生变化后，就调用IGenerator来创建代码，在这个地方使用IJvmModelInffer、IJvmAssociator等手段，来处理dsl文件。

IScopeProvider一个用途是在resolve过程中使用的。

在BuilderPartcipant中还需要维护所谓的索引。

在JvmTypes中建立了java类和JvmType之间映射关系。保证dsl中能够使用到java类的元素。

在dsl的resource中插入了JvmType的对象，在需要输出为Scope的对象时，是如何输出jvmType对象，而不是dsl元素的呢？

# Xtext

Xtext relies on the Antlr parser generator for the parsing of input files. On top of that the framework provides lot's of added value such as strongly typed ASTs, abstractions for linking and static analysis as well as IDE integration for Eclipse.

For that purpose, Xtext generates two Antlr grammars. One for the production parsing where the actual AST is produced, and a second grammar that is used to consume events to compute the content proposals for the Eclipse editor.

为了那个目标，xtext生成两个anltr语法。一个用来产生解析创建抽象语法树，第二个用来消费事件为了编辑器的内容建议。

既然自己还大量使用，居然标识Deprecated，意思是告诉我们不要再使用了，要使用新版本提供的机制才行，但是毕竟很奇怪，在自己的代码中也应该体现出来采用了新的实现才对。

有几个战场

1. 围绕Scope的；

# 2月19

从儿子出生就唱小燕子哄他睡觉，最近，只要一唱小燕子，儿子就咿咿的抗议，问他怎么了，他就说一居听不懂的话，昨天晚上睡觉，又给儿子唱小燕子，儿子抗议后，问他，他说‘马马’，原来最近总是给他讲小马过河，他要听小马过河的故事。

# 线索

在xtend的基础上试着尝试js的generator。

问题：当workspace中存在其他已经dsl资源，当前编辑器如果import了其他dsl资源，此时这些资源需要通过JvmModelInffer来把自己变成JvmModel，这个工作的是如何完成的？

是否还是通过Builder呢，这样的话，就需要在当前文件变化后，把导入的资源也generator才行吧。

呵呵，复杂了，估计是这样的，如果dsl生成了java类，此时就可以通过java类映射JvmType模型的方式来进行了吧，这样的话，意味着如果dsl存在错误的话，是否不进行generator呢？

呵呵，应该是通过这个机制，对于xtend来说，即使xtend文件存在错误，generator也会被调用，来生成带有错误的java类。

经过实验可以看出，确实是通过生成的java类来进行的。

如果没有保存，其他dsl中的信息好像没有映射为jvmModel了。

清理一下大概的脉络：

JvmModelInfer：这个接口出现在xbase插件中，意味着需要与jvmType转换才会需要的。因此在xtext中没有关系。

IGenerator：xtext中的接口，意味着是dsl的基础设施之一，在Builder中会被调用。默认实现是什么也不干；

在XtextSource的一个子类中：

# EMF

eDeliver这个属性的意思是是否把变化交付给侦听器。

# State

这个是Builder中的一个概念，Builder的一个主要工作，当dsl文件发生变化后，尤其是增加、删除resource后，维护一个全局的resourceset内容。

# JsDSL

如果这个dsl直接采取JvmModel，也就意味着不需要JvmModelInfer了。由于这个里面也不涉及java的内容，自然也不需要映射java类的机制。

这样的话，这个dsl就可以得到很大的简化了。

需要IGenerator，在Builder的过程中，编译出来js文件；由于可以借鉴xbase的代码，compiler部分的，因此这个部分的深度有保障。

这样的话，最大的工程就是Scope的处理了。

# Xtend Generator

奇怪，在生成java代码的地方怎么没有看到函数内容被生成呢？

在Infer时，其实可以不用，到了Generator中才需要。

呵呵，又一次看到了Adapter在EMF中的威力，可以这一次感觉使用的并不好。

在JvmOperation中——映射java方法，并没有函数体的相关信息，到了xtend中，在处理xtend的xtendFunction时，就把body（XBlockExpression）通过Adapter设置到了JvmType所在资源中了。于是到了要Generator时，就要绕一个大圈子，来获取这个body，然后生成需要的代码了。当然函数体的代码可以到真正需要时再生成好了。

这个地方之所以别扭，可能是有一个问题：就是Containment的问题。首先这个表达式位于XtendFunction的expression属性上，应该是以Containment的形式存在的。

到了JvmOperation中，如果这个属性以Containment形式存在于JvmOperation中的话，必须要从XtendFunction的expression复制一个，否则，就不能以Containment的形式了，而如果不已Containemnt的形式存在的话，可能会有其他的问题。

那么xtend为什么不直接采用JvmType的模型来设置自己的语法文件呢？

# Def create

<https://groups.google.com/forum/#!topic/xtend-lang/15gz5CoqCAI>

# 2月20

到了TypesSystem，这个是xtext中最难的一部分。由于缺少这个部分的背景知识。一下子不得其门完全是正常的。

在js中，不允许方法重载，应该会简化很多。

还需要仔细思考接口在js中的问题；

Js中运算时的类型推导比较其他语言应该是最简单的了；

# If

XIfExpression **returns** *XExpression*:

{*XIfExpression*}

'if' '(' if=XExpression ')'

then=XExpression

(=>'else' else=XExpression)?;

这个表达式很优美。=>保证了else总是匹配最近的if。

XAssignment是从XFeatureCall继承的；

# 一个方案

运行xtend，然后调试一下看看。

嗯，TypesSystem部分总算是找到点头绪。

从XBase看出，TypeSystem还真是一门语言的核心特性之一。

你面融合了Scope的处理、类型适合性、类型计算等。

采用一个State的概念，并且在表达式部分，这个State还是嵌套的，从而组成一颗State的树。在进行类型计算过程中，对于VariableDeclaration顺便被添加到了Scope上。

好像还有一个概念，就是ScopeSession的概念，还需要仔细体会才行。

几个类：

IBatchTypeResolver：这个接口类似于门面接口，封装了类型系统的细节。

对于每一个对象都调用IBatchTypeResolver的resolveTypes，其返回结果是IResolvedTypes。

ILinkingCandidate：link的候选者。

# <=> 太空船运算符

Groovy在语言级别提供了很多有意思的运算符。 其中有一个叫做太空船(spaceship)运算符。 之所以叫他太空船运算符，是因为它的语法是 <=> 看上去有点像UFO。 这个运算符是Comparable接口的compareTo方法的另一个表现方式。这就是说只要你在自己的类里实现了compareTo方法，就是可以在你的代码里使用 <=>了。当然已经实现了Comparable接口的其他类也可以使用这个运算符。

Java代码



class Person implements Comparable {

String username

String email

int compareTo(other) {

this.username <=> other.username

}

}

assert -1 == ('a' <=> 'b')

assert 0 == (42 <=> 42)

assert -1 == (new Person([username:'mrhaki', email: 'test@email.com']) <=> new Person([username:'zavaria', email:'tester@email.com']))

assert [1, 2, 3, 4] == [4, 2, 1, 3].sort{ a, b -> a <=> b }

可以简化为compareTo方法的简写操作符。

# [Groovy: Elvis运算符](http://zjumty.iteye.com/blog/1882380)

博客分类：

Groovy里有一种操作符叫Elvis运算符，可以简化Java里的二元操作符。如果我们需要在某个值是空或false时指定一个默认值时，使用Elvis运算符会使表达式更简洁。这里的false判断遵循Groovy的Truth判定方法：把null，空字符串，空集合视为false。   
  
例如：

Java代码



def sampleText

// Java里的二元运算符（Groovy里也通用）.

def ternaryOutput = (sampleText != null) ? sampleText : 'Hello Groovy!'

// 下面是使用Elvis运算符的例子， 意思是: 如果sampleText不为空会空字符串，

// 则把sampleText赋给elvisOutput，否则把'Viva Las Vegas!'赋给elvisOutput.

def elvisOutput = sampleText ?: 'Viva Las Vegas!'

至于为什么叫Elvis运算符，是因为'?:'跟一个叫Elvis的摇滚明星(猫王)的发型很像。

# With Operator

The with operator is very handy when you want to initialize objects or when you want to use a particular instance a couple of time in subsequent lines of code. It simply passes the left hand side argument to the lambda on the right hand and returns the left hand after that.

Here's an example:

val person = new Person => [

firstName = ’Homer’

lastName = ’Simpson’

address = new Address => [

street = ’742 Evergreen Terrace’

city = ’SpringField’

]

]

# Xtext

这一团乱麻好像快被解开了，曹永猛同志，加油啊。

一切都从Resource开始，从getResource、getObject开始，或者说魔力就来自于这两个方法的使用。

# 一个问题

用IDL语言编写rpc和bean，然后编译为java类，和jsDSL，也就可以可。

在编译为jsdsl的过程中，需要携带json相关的代码，在java的代码中，需要携带json序列化相关的代码。

# Enum

在jsdsl中不能使用enum，因为enum按照java来看首先是一个类型，符合作为类型的基本特性。而js中是不支持这个类型的，如果要支持enum，就意味着在js中，这个枚举不能再当被成整数来使用了。

或者意味着

# Xtend

为什么xtend不使用直接使用JvmType，又找到一个原因。就是在设置modifiers时，如果采用unorder group的话，语法上不行。因此就只能采用modifiers= CommonModifer\*的形式了。

# 2月21

Xbase中大量使用了xbase的emf对象，而emf对象需要依赖common.types定义的EMF对象，如果在jvmType中直接定义方法的body的话，意味着需要使用xexpression，而这会造成循环依赖，不好处理了。

其实，如果没有xbase的要把JvmTypes分离出来的责任，可以把这二者合二为一，从而也就不会出现插件的循环依赖，而且模型上也比较单一了吧。

不再需要JvmodelInferred了，什么JvmAssociation之类也就需要需要了。

其实本来就应该在一起，一个是类型定义、一个是表达式定义，二者耦合的很紧，如果非要分开，自然需要付出代价。

# 合并到一起

把xbase、types、annonation合并到一起。需要先把现在的备份一下。

标注在jsdsl中有用吗，应该是有用的。一种场景：一个依赖属性不允许binding，这个信息如何设置呢？如果设置到依赖属性上，这个是运行时信息，在编译阶段获取不到，此时就只有通过标注来实现了。这个标注是可以在编译阶段来使用的。

呵呵，看样子是没有办法绕过去的了。

# Annonation的语法

Annotations: Annotation [Annotations]

Annotation: @ TypeName [( [Identifier =] ElementValue)]

ElementValue: ConditionalExpression Annotation ElementValueArrayInitializer

# 3月2

晓霖今天打电话，说是最近要到广州。儿子14岁了。

# 3月4

讨厌的wps，居然把这段时间的记录搞丢了。

**没有语法检查了**

可能的原因是什么呢？

# 3月5

继续调试合并后的ss

看到了NumberLiterals的处理，还需要看看booleanListral的处理。

BooleanLiteral是通过Boolean.class爱处理的，也就是说，是通过java的class来处理的。这个地方需要修正，可以直接返回Boolean的Type即可。

对于NumberLiteral的处理也是类似，应该不需要绕java的class的方式，可以直接处理。

在Xtend中，还有的Literal有：

StringLiteral

BooleanLiteral

CharacterLiteral

ListLiteral

NumberLiteral

ArrayLiteral：就是ListLiteral等价产物。

NullLiteral：换算成AnyTypeReference的形式了。

Object的处理：

这个地方的一个关键是：typesystem下的references了。

# Types

synonym

# Xtext2.6

看样子等到xtext2.6出来后，还需要第三次对xtend进行私有化处理。

因为有一些bug被解决了。

# XAbstractFeratureCall

有一个feature属性。

在BinaryOperation中，feature是运算符；

UnaryOperation：feature是一元运算符

在MemberFeatureCall中呢：成员属性

# 一个经验

如果没有设置 -XX:MaxPermSize这个参数的话，好像很容易就会出现outofheapmemory这个异常了。设置了之后，好像就没有再出现这个错误，而且Ss IDE感觉稳定了很多。

# 3月6

Ss IDE终于算是正常了。

# Xtend.lib

这个库到底该如何理解呢？

在编译xtend类文件时，需要xtend.lib被设置到classpath上，此时的机制仅仅是需要运行xtend编译环境需要，而不是当前编辑的xtend源文件的内容需要，从这个角度看，此时源文件中应该仅仅是dsl.xbase.lib和ss.lib才对。

当然如果编译xtend类文件自然需要添加xtend2.lib到当前的工程以来中了。

呵呵，这个理解应该才是正确的。这个地方它饶了。

也不是这么绝对，在xtend源文件中，使用的有一些lib，就需要区分，例如：如果使用了procedures之类的话，而这个又需要与编译为java后的代码一致，此时这个procedures就应该是xtext.xbase.lib中的了。

在类型判断部分，此时的procedures应该是dsl.xbase.lib中的。

# 总结

在迁移过程中有如下的教训：

1. 确实可以把xtend、xbase、xannotation、xtype的emf模型合并到一起，这样折腾是需要在jvmType中直接潜入XExpression作为方法的内容；
2. Xtext.ui.shared的作用是注册Builder的扩展点，注册了这个Builder后，才会对ss的源文件进行构造。同时提供sharedModule；
3. Xtend.lib、xtext.xbase.lib应该不会出现在移植后的源代码中，应该需要的是dsl.xbase.lib和ss.lib；当然有些特殊情况下，还是需要xtend.lib和xtext.xbase.lib的，因为编译后的java源代码的需要；
4. Heap空间太小会导致很多奇怪的问题，例如：application直接退出了；

# 为什么Plugins中的%DslBuilder.name找不到

今天遇到问题就是Plugin.properties放在plugin.xml的相同目录下, Eclipse里面看国际化也是非常正确, 但是运行以后就是只显示"%XXXX"这样的字符串, 如同没有国际化一样. 一直没有搞明白原因, 到网上查了一下, 终于找到答案.

需要**Bundle-Localization**: plugin的设置。

# 移植XAML到新的SsIDE中