Windows 8 开发 31 日

第 06 日

搜索合约

译者：BeyondVincent(破船)

时间：2013.4.23

版本： 2.0

关于破船

程序猿砌墙于云南昆明!

长期扎根移动软件开发!

爱跑步爱打篮球爱运动!

命中无大富大贵之面相!

愿健康与平淡相随一生!

你可以发邮件与破船取得联系: BeyondVincent@gmail.com

还可以关注破船的微博: 腾讯微博和新浪微博。

这里是破船的个人博客，欢迎光临：破船之家

关于 Windows 8 开发 31 日翻译

Windows 8 开发 31 日是由 Jeff Blankenburg 和 Clark Sell 原创的。

官方站点：http://31daysofwindows8.com/

涉及到两个版本：

XAML/C#（由 Jeff Blankenburg 撰写）

HTML5/JS （由 Clark Sell 撰写）

其中涉及到的资源和相关代码请到这里下载：

https://github.com/csell5/31DaysOfWindows8

在这里，由于破船对 HTML5/JS 不熟悉，所以只翻译 XAML/C#相关主题。

建议大家前往看原创内容，如果看不明白，再来这里看我翻译的相关内容。

如果翻译不正确的地方，可以通过上面的联系方式告诉破船。

破船祝你阅读愉快！

目录

关于破船 2

关于 Windows 8 开发 31 日翻译 3

目录 4

第 06 日搜索合约 5

1.0. 介绍 5
1.1. 开始 6
1.2. 搜索哲理（searchphilosophy） 9
1.3. 让搜索开始工作 12
1.4. 定义搜索建议 18
1.5. 当检测到键盘输入时强制进行搜索 20
1.6. 声明 20
1.7. 总结 22

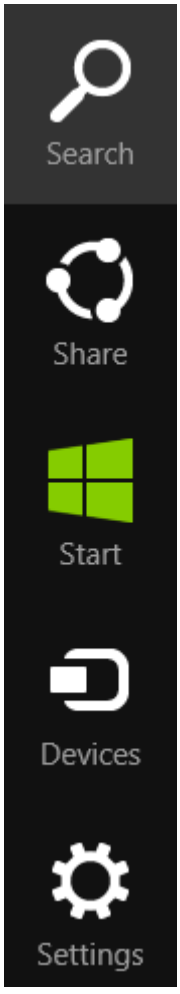
第 06 日搜索合约

1.0. 介绍

昨天我介绍了如何在程序中添加设置合约。今天将介绍搜索合约，明天则是共享合约。搜索和共享是两个非常感实用的合约，因为即使程序没有在运行，这两个合约都会让你的程序可以使用，也就是说你的程序可以潜在的以另外的一种方式暴露给用户。

在过去的 10 年里面，”搜索”已经典型的成为搜索引擎的代名词。几年前，搜索在 Windows 实际上已经变成了主流，如果你跟我一样，也会变得非常习惯的点击搜索按钮，然后输入程序名字，而不是去亲自寻找程序。今天，在 Windows 8 中，你也可以输入搜索内容，并且搜索到结果会很友好的显示出来。

更好的是，在 Windows 8 中，已经扩展了搜索，可以将搜索添加到我们的 Windows Store Apps 中。现在，搜索不再神秘，我们只需要响应很少的几个事件就可以给用户显示正确的内容。搜索已经



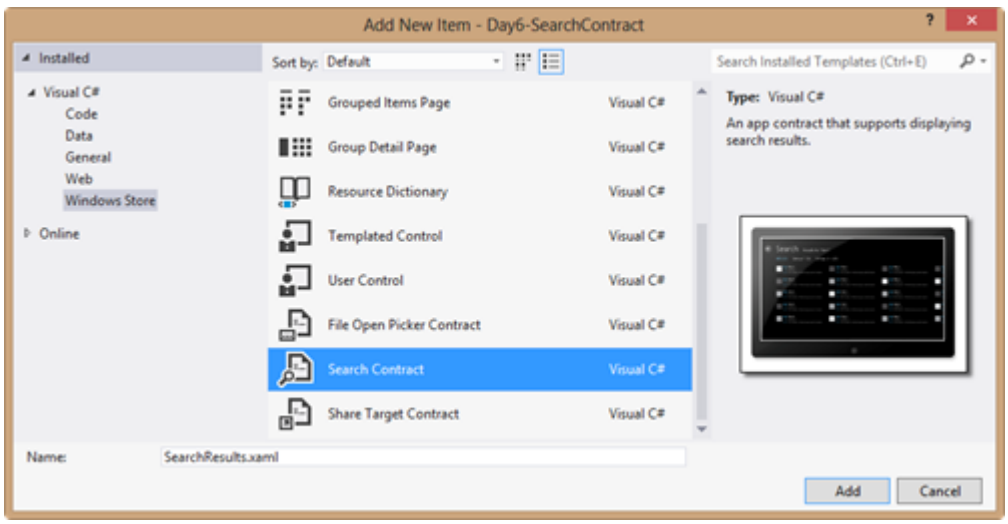
改变了典型的从“入口点”（entry point）进入程序。现在，用户不需要打开程序，然后再找到搜索框进行搜索，而是可以通过搜索面板，输入内容，然后直接在内进行程序查找。

1.1. 开始

正如本系列文章中，每天所做的步骤，从 Blank App template 开始，本文我也是从这里开始。从 Blank App template 创建工程，可以避免复杂而对目前来说是无用的代码。

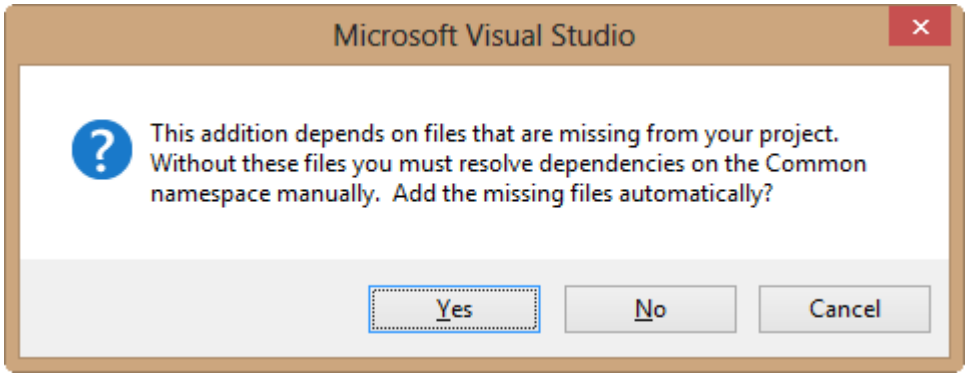
首先创建一个新的工程（然后替换默认的图片…你知道怎么做吗？在第 01 日里面有介绍，[你也可以在这里下载到相关的图片]）现在，给程序添加一个搜索。非常方便的是基本的搜索添加只需要一路 click 即可。

在工程中单击右键,选择 Add->New Item…>Search Contract

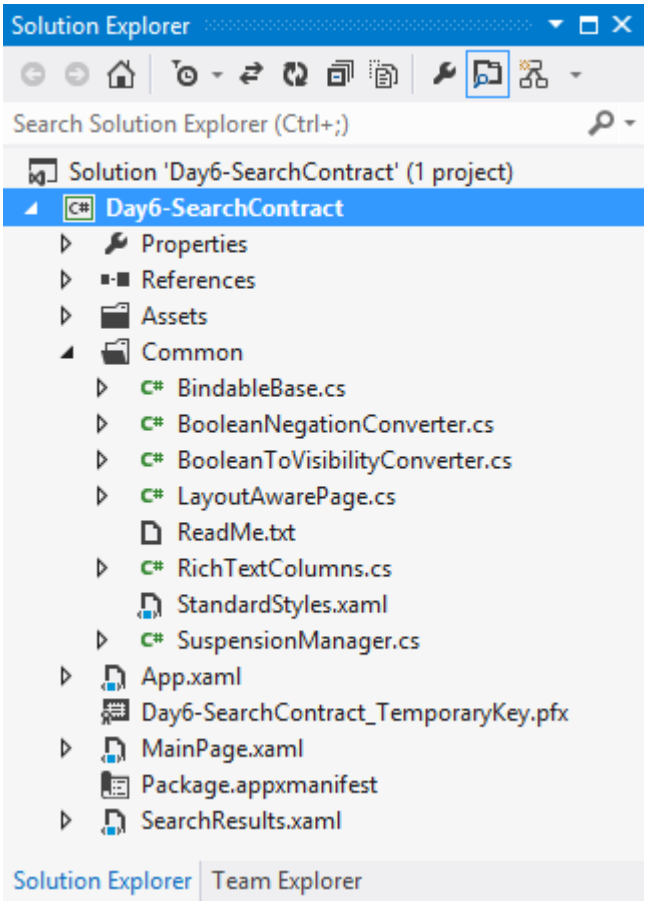


在上面图中要求输入搜索页面的名称，我使用 SearchResults.xaml，你可以使

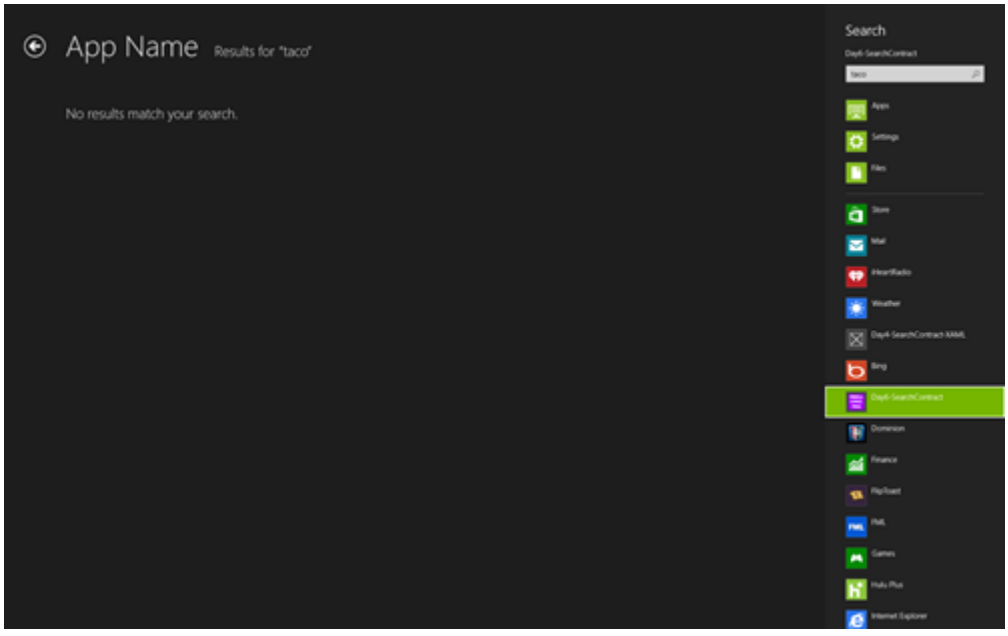
用你想要的名称。当你点击“Add”按钮后，会有提示添加一些附属的依赖文件到工程中。



稍后，我会简短的介绍添加的每一个文件。现在，你只需要知道工程解决方案看起来像下面这样：



这时，如果你运行工程，然后选择 Search charm，并输入内容,会看到如下画面。（我输入的是 taco）



如上面这个图，程序运行起来，搜索画面也出来了，但是由于我还没有对搜索进行编码处理，所以这里没有提供任何的数据。在进行编码处理之前，我先来谈谈关于 Windows 8 搜索背后的哲理。

1.2. 搜索哲理（searchphilosophy）

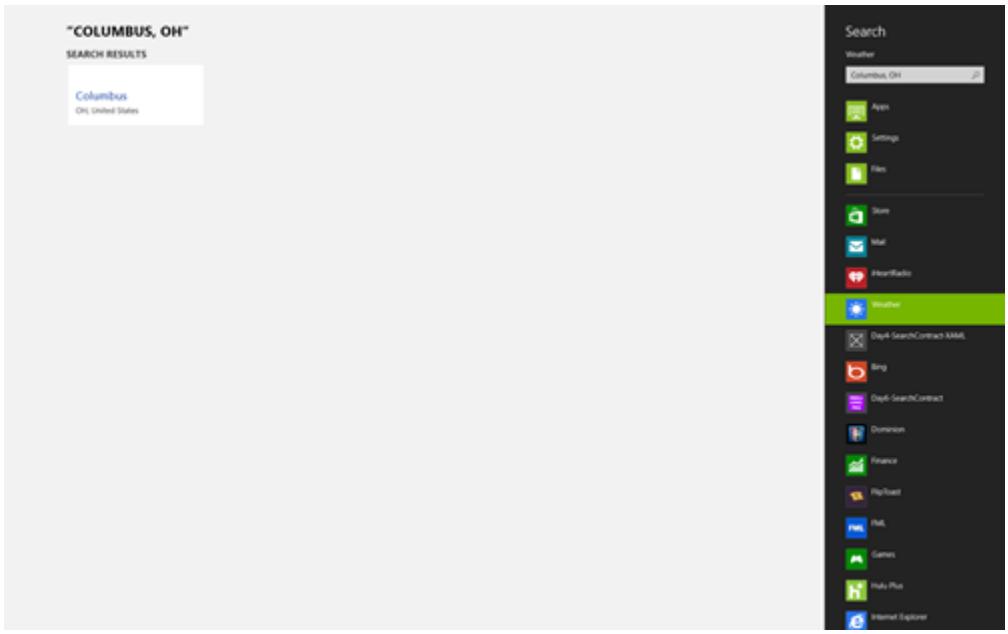
当你在打量 Windows 8 程序时，会发现每个程序涉及的功能很很少。你不会找到 Outlook 这样管理许多类型数据的程序。而替代的是 Calendar 程序，Mail 程序，People 程序。

Windows 8 程序建议一个程序只做一件事情，并做好一件事情。希望你写的程序是这样的。当 Windows 8 程序没有在运行的时候，同样可以进行搜索。这就意味着，当用户搜索一些内容时，你的程序应该立即将某些数据展现给用户。

例如，iHeartRadio 程序。如果我用它进行搜索，我希望搜索的是一个 radio

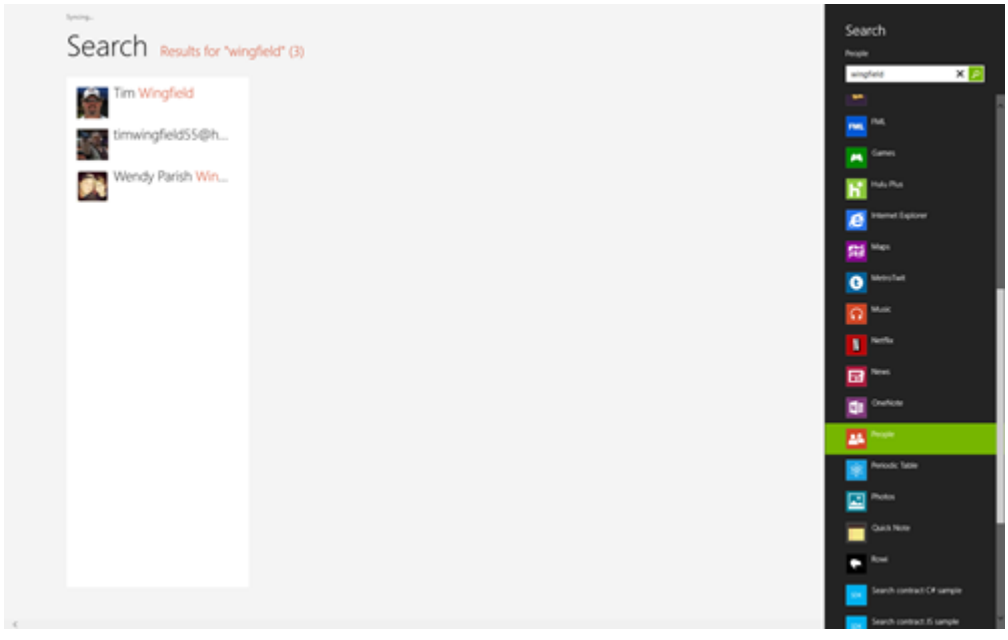
station。

如果是天气程序，我希望搜索某个城市。



Netflix ，我可能希望搜索的是电影的标题。

最后，如果是 People 程序，我希望搜索的是人名。



因此，当你在开发程序时，问自己一个问题：

“用户想要在我的程序中搜索什么？”

另外一个需要重点讨论的是关于围绕搜索方案的概念“Search”和“Find”。搜索 charm 不应该用于在一个文档内部进行搜索。在你的程序中，搜索应该总是可以处理某些事情，即使没有在运行状态。

这里是微软给出的相关指南：如何使用搜索 Charm

1.3. 让搜索开始工作

在之前添加的 Search 合约到工程中只是第一步。下一步，需要提供一些有价值的数据供用户搜索。你还记得吗，在第 04 日的文章，我从元素周期表中创建了一小组数据来演示 SemanticZoom 控件。

在这里的示例中，我们可以使用 04 日中的数据。如果你没有看过，那么下面我给出 Element 类的代码：

```
namespace Day6_SearchContract
{
classElement
    {
publicdouble AtomicWeight { get; set; }
publicint AtomicNumber { get; set; }
publicstring Symbol { get; set; }
publicstring Name { get; set; }
publicstring Category { get; set; }
publicstring State { get; set; }
    }
}
```

为了简单起见，我在 SearchResults.xaml.cs 文件中填充了一个 List<Element>。我创建了一个新的方法 BuildElementList()，并在构造函数中调用该方法。

```
        List<Element> elements = newList<Element>();
```

```
public SearchResults()
        {
this.InitializeComponent();
            BuildElementList();
        }

privatevoid BuildElementList()
        {
            elements.Add(newElement { AtomicNumber = 1, AtomicWeight = 1.01, Category = "Alkali Metals",
Name = "Hydrogen", Symbol = "H", State = "Gas" });
            elements.Add(newElement { AtomicNumber = 2, AtomicWeight = 4.003, Category = "Noble Gases",
Name = "Helium", Symbol = "He", State = "Gas" });
            elements.Add(newElement { AtomicNumber = 3, AtomicWeight = 6.94, Category = "Alkali Metals",
Name = "Lithium", Symbol = "Li", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 4, AtomicWeight = 9.01, Category = "Alkaline Earth
Metals", Name = "Beryllium", Symbol = "Be", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 5, AtomicWeight = 10.81, Category = "Non Metals",
Name = "Boron", Symbol = "B", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 6, AtomicWeight = 12.01, Category = "Non Metals",
Name = "Carbon", Symbol = "C", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 7, AtomicWeight = 14.01, Category = "Non Metals",
Name = "Nitrogen", Symbol = "N", State = "Gas" });
            elements.Add(newElement { AtomicNumber = 8, AtomicWeight = 15.999, Category = "Non Metals",
Name = "Oxygen", Symbol = "O", State = "Gas" });
            elements.Add(newElement { AtomicNumber = 9, AtomicWeight = 18.998, Category = "Non Metals",
Name = "Fluorine", Symbol = "F", State = "Gas" });
            elements.Add(newElement { AtomicNumber = 10, AtomicWeight = 20.18, Category = "Noble Gases",
Name = "Neon", Symbol = "Ne", State = "Gas" });
            elements.Add(newElement { AtomicNumber = 11, AtomicWeight = 22.99, Category = "Alkali
Metals", Name = "Sodium", Symbol = "Na", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 12, AtomicWeight = 24.31, Category = "Alkaline Earth
Metals", Name = "Magnesium", Symbol = "Mg", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 13, AtomicWeight = 26.98, Category = "Other
Metals", Name = "Aluminum", Symbol = "Al", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 14, AtomicWeight = 28.09, Category = "Non Metals",
Name = "Silicon", Symbol = "Si", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 15, AtomicWeight = 30.97, Category = "Non Metals",
Name = "Phosphorus", Symbol = "P", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 16, AtomicWeight = 32.06, Category = "Non Metals",
Name = "Sulfur", Symbol = "S", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 17, AtomicWeight = 35.45, Category = "Non Metals",
Name = "Chlorine", Symbol = "Cl", State = "Gas" });
            elements.Add(newElement { AtomicNumber = 18, AtomicWeight = 39.95, Category = "Noble Gases",
Name = "Argon", Symbol = "Ar", State = "Gas" });
            elements.Add(newElement { AtomicNumber = 19, AtomicWeight = 39.10, Category = "Alkali
Metals", Name = "Potassium", Symbol = "K", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 20, AtomicWeight = 40.08, Category = "Alkaline Earth
Metals", Name = "Calcium", Symbol = "Ca", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 21, AtomicWeight = 44.96, Category = "Transitional
Metals", Name = "Scandium", Symbol = "Sc", State = "Solid" });
            elements.Add(newElement { AtomicNumber = 22, AtomicWeight = 47.90, Category = "Transitional
```

```
Metals", Name = "Titanium", Symbol = "Ti", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 23, AtomicWeight = 50.94, Category = "Transitional
Metals", Name = "Vanadium", Symbol = "V", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 24, AtomicWeight = 51.996, Category = "Transitional
Metals", Name = "Chromium", Symbol = "Cr", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 25, AtomicWeight = 54.94, Category = "Transitional
Metals", Name = "Manganese", Symbol = "Mn", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 26, AtomicWeight = 55.85, Category = "Transitional
Metals", Name = "Iron", Symbol = "Fe", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 27, AtomicWeight = 58.93, Category = "Transitional
Metals", Name = "Cobalt", Symbol = "Co", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 28, AtomicWeight = 58.70, Category = "Transitional
Metals", Name = "Nickel", Symbol = "Ni", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 29, AtomicWeight = 63.55, Category = "Transitional
Metals", Name = "Copper", Symbol = "Cu", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 30, AtomicWeight = 65.37, Category = "Transitional
Metals", Name = "Zinc", Symbol = "Zn", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 31, AtomicWeight = 69.72, Category = "Other
Metals", Name = "Gallium", Symbol = "Ga", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 32, AtomicWeight = 72.59, Category = "Other
Metals", Name = "Germanium", Symbol = "Ge", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 33, AtomicWeight = 74.92, Category = "Non Metals",
Name = "Arsenic", Symbol = "As", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 34, AtomicWeight = 78.96, Category = "Non Metals",
Name = "Selenium", Symbol = "Se", State = "Solid" });
                elements.Add(newElement { AtomicNumber = 35, AtomicWeight = 79.90, Category = "Non Metals",
Name = "Bromine", Symbol = "Br", State = "Liquid" });
                elements.Add(newElement { AtomicNumber = 36, AtomicWeight = 83.80, Category = "Noble Gases",
Name = "Krypton", Symbol = "Kr", State = "Gas" });
            }
```

显然，你希望将你的数据放在别的地方，不过这篇文章是关于搜索的，不是关于 data sources 的。

如果你查看 SearchResults.xaml.cs 文件中默认的方法 Filter_SelectionChanged()，你会看到如下注释掉的内容：

```
// TODO: Respond to the change in active filter by setting
this.DefaultViewModel["Results"]// to a collection of items with bindable Image, Title,
Subtitle, and Description properties
```

在这段注释下面，我将添加我的逻辑处理——从我的数据源中获取到适当的

记录。不过在这之前，该方法中，有许多假设，在写代码之前，弄明白这些假设非常重要。

第一种假设就是你的搜索结果是一个对象集合，每一个对象都有一个 Image、Title、SubTitle 和 Description 属性。当然你的数据中不会有这些属性。你也不必强制将这些属性添加到你的数据中。这种假设的原因很简单。

打开你的 SearchResults.xaml 页面。找到名为“resultsGridView”的 GridView 控件，以及名为“resultsListView”的 ListView 控件。这种类型的控件我在第 04 日已经介绍过了。这里的两个控件都已经各自有一个 ItemTemplate 了。它们使用默认的 DataTemplate，名称为“StandardSmallIcon300x70ItemTemplate”和“StandardSmallIcon70ItemTemplate”。在 Common/StandardStyles.xaml 文件中，可以找到这两个 DataTemplate。它们都绑定到上面说到的 4 个属性。我并不想使用这些模板，但它们是好的开始。

我下面所做的就是将“StandardSmallIcon300x70ItemTemplate”整个 entry 拷贝到 SearchResults.xaml 页面的 Page.Resources 中。下面是我修改过的内容：

```
<Page.Resources>
<CollectionViewSource x:Name="resultsViewSource" Source="{Binding Results}"/>
<CollectionViewSource x:Name="filtersViewSource" Source="{Binding Filters}"/>
<common:BooleanToVisibilityConverter x:Key="BooleanToVisibilityConverter"/>
<!-- TODO: Update the following string to be the name of your app -->
<x:String x:Key="AppName">Search Contract Example</x:String>
<DataTemplate x:Key="ModifiedSmallIcon300x70ItemTemplate">
<Grid Width="294" Margin="6">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Border Background="{StaticResource ListViewItemPlaceholderBackgroundThemeBrush}" Margin="0,0,0,10"
Width="40" Height="40">
<TextBlock Text="{Binding Symbol}" FontSize="32" FontWeight="Bold" TextAlignment="Center"
```

```
VerticalAlignment="Center" />
</Border>
<StackPanel Grid.Column="1" Margin="10,-10,0,0">
<TextBlock Text="{Binding Name}" Style="{StaticResource BodyTextStyle}" TextWrapping="NoWrap"/>
<TextBlock Text="{Binding Category}" Style="{StaticResource BodyTextStyle}" Foreground="{StaticResource
ApplicationSecondaryForegroundThemeBrush}" TextWrapping="NoWrap"/>
<TextBlock Text="{Binding State}" Style="{StaticResource BodyTextStyle}" Foreground="{StaticResource
ApplicationSecondaryForegroundThemeBrush}" TextWrapping="NoWrap"/>
</StackPanel>
</Grid>
</DataTemplate>
</Page.Resources>
```

可以看出，跟原来的内容几乎相同，只不过名字修改为“ModifiedSmallIcon300x70ItemTemplate”。另外不同的地方是我将 Image 控件修改为 TextBlock，以及将绑定的对象修改为我的对象所拥有的属性：Symbol，Name，Category 和 State。同时，我将 GridView 和 ListView 的 ItemTemplate 修改为我修改过的版本 ModifiedSmallIcon300x70ItemTemplate。现在，可以开始给 SearchResults 页面提供一些结果了。

还记得之前说过的注释吗？在 Filter_SelectionChanged()方法中,我们再到该方法中，添加一些代码：

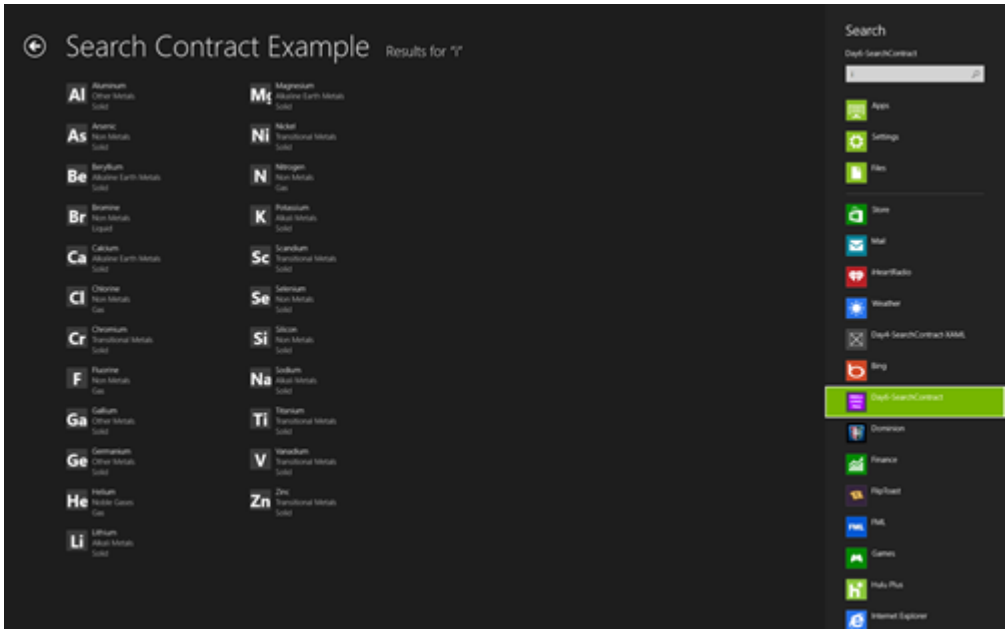
```
IEnumerable<Element> searchResults = from el in elements
where el.Name.ToLower().Contains(searchString)
orderby el.Name ascending
select el;

this.DefaultViewModel["Results"] = searchResults;

// Ensure results are found
object results;
IEnumerable<Element> resultsCollection;

if (this.DefaultViewModel.TryGetValue("Results", out results) &&
      (resultsCollection = results asIEnumerable<Element>) != null&&
      resultsCollection.Count() != 0)
{
VisualStateManager.GoToState(this, "ResultsFound", true);
return;
```

```
}
```

我添加了一个简单的 LINQ 语句从 data source 中获取适当的数据，然后将结果赋值给 this.DefaultViewModel[“Results”]。这些结果将要填充在 GridView 控件中。

在该函数后面的代码中，我也做了一些修改——使用一个关于 Element 对象的 IEnumerable 集合。如果根据搜索词确实搜索到了一些结果，那么将会显示出来。

现在，运行程序进行搜索，可以看到如下界面：



你也可以将程序关闭，然后在机器上执行搜索，并选择你的程序。它的工作方式跟运行着的时候一样。

在此，如果你认为已经完成搜索任务了。那么你错了。我还会介绍更多关于搜索的内容。特别是如何从程序中匹配出用户输入的内容，然后给用户做出提示。下面来看看吧。

1.4. 定义搜索建议

跟上面已经完成的内容相比,定义搜索建议实际上很简单。同样在 SearchResults.xaml 页面中，需要从 SearchPane（在名称空间 Windows.ApplicationModel.Search 中）开始。在构造函数中实例化一个 SearchPane。

```
SearchPane searchPane;
public SearchResults()
{
this.InitializeComponent();
        BuildElementList();
        searchPane = SearchPane.GetForCurrentView();
}
```

上面这步完成，下一步是创建一个给 SuggestionsRequested 时间创建一个 event handler。创建一个新的方法 OnNavigatedTo，并添加一个新的 event handler，如下代码。（同样的，创建一个 OnNavigatingFrom 来移除该 event handler）。

```
protectedoverridevoid OnNavigatedTo(NavigationEventArgs e)
{
base.OnNavigatedTo(e);
        searchPane.SuggestionsRequested += searchPane_SuggestionsRequested;
}
```

```
protectedoverridevoid OnNavigatingFrom(NavigatingCancelEventArgs e)
{
base.OnNavigatingFrom(e);
        searchPane.SuggestionsRequested -= searchPane_SuggestionsRequested;
}
```

```
void searchPane_SuggestionsRequested(SearchPane sender, SearchPaneSuggestionsRequestedEventArgs args)
{
        args.Request.SearchSuggestionCollection.AppendQuerySuggestions((from el in elements
where el.Name.ToLower().StartsWith(args.QueryText.ToLower()) ||
```

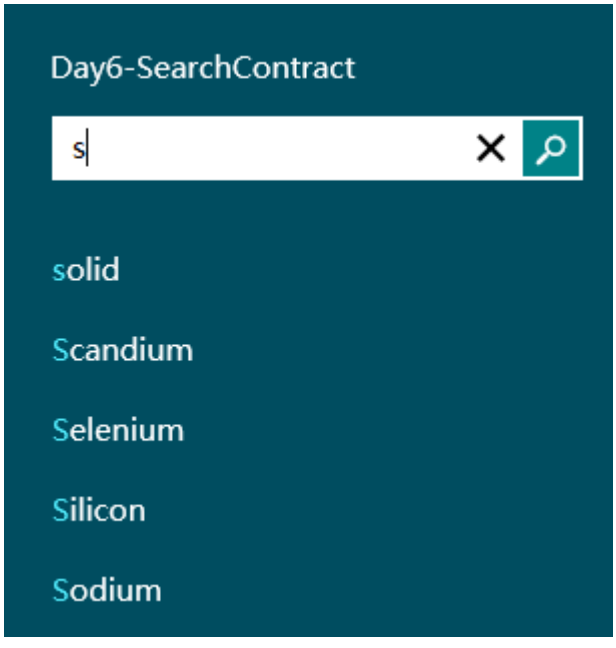
```
el.Symbol.ToLower().StartsWith(args.QueryText.ToLower())
orderby el.Name ascending
select el.Name).Take(5));
}
```

上面的代码可以看到,我提供了一个 SearchSuggestionCollection，选取最多 5 个字符串作为建议。这也是系统限制的，最多只能有 5 个。

好了,就是上面这些工作即可。现在运行程序，并搜索元素，如果你输入的字符与元素名或者 symbol 匹配，则会看到最多 5 个搜索建议项。

如下图所示:

下图是译者所加,原文中没有提供此图:



1.5. 当检测到键盘输入时强制进行搜索

这再说一点小技巧，可以使用户在输入时简单的打开搜索面板。只需要在 App.xaml.cs 文件的 OnLaunched 和 OnSuspending 方法中各添加一行代码，就可以在整个程序中实现这个功能。

在 OnLaunched 方法中添加如下代码：

```
SearchPane.GetForCurrentView().ShowOnKeyboardInput = true;
```

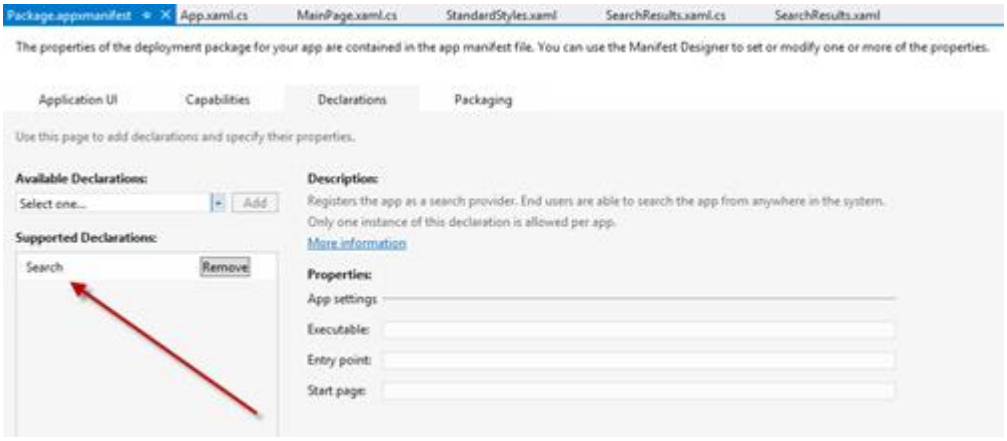
当程序关闭时，在 Onsuspending 方法中添加如下代码，可以取消该功能：

```
SearchPane.GetForCurrentView().ShowOnKeyboardInput = false;
```

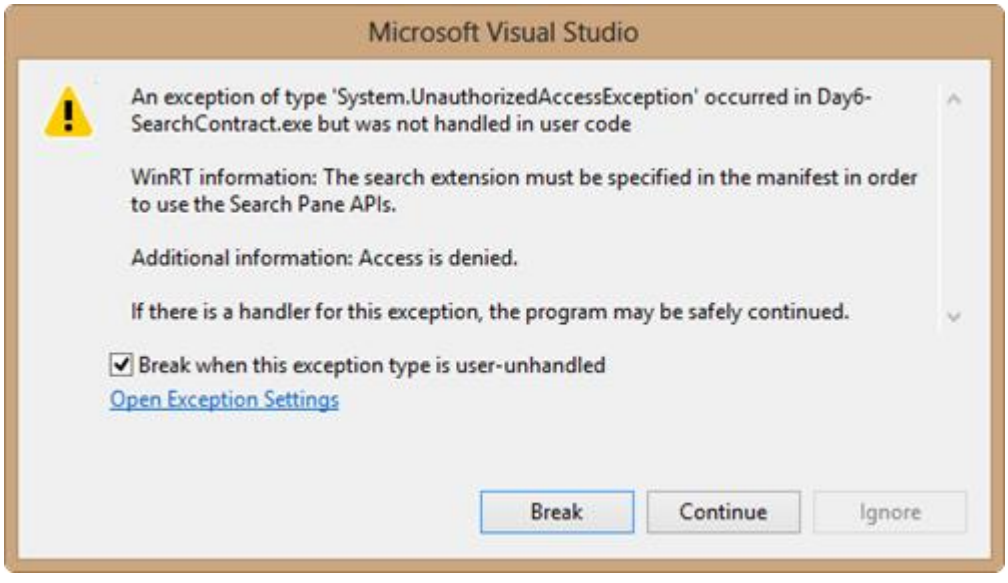
现在,运行程序,只需要输入内容,Search Charm 会自动打开,并开始捕获你的输入内容。

1.6. 声明

这是本文介绍的最后一个内容。这主要是让你了解有这么回事，实际上不需要你亲自去做的。当你把搜索合约添加到工程时，这里有好几个文件被添加到工程中，并且在你的 appmanifest 文件中也添加了相关声明。如果你打开这个 appmanifest 文件,并导航到声明选项，你会看到如下内容：



如果没有这个声明,我们添加的搜索功能将不会被允许执行.并且会得到一个访问被拒绝的异常提示。



需要记住，在程序中使用搜索相关的 APIs 之前，需要对搜索进行声明。

1.7. 总结

今天,我们快速的学习了如果在程序中添加搜索合约.搜索合约为程序提供了一种新的方法——将你的程序暴露给用户。现在，你只需要知道，你想暴露的确切内容。

点击下面的按钮,下载完整的示例解决方案:

明天,我们将学习本系列中涉及到的最后一个合约——共享合约。到时候见！

感谢你的阅读！

如果对这篇文章有什么想法想法，可以与破船联系，破船的联系方式在文章开头。

破船