

# TL技术交流

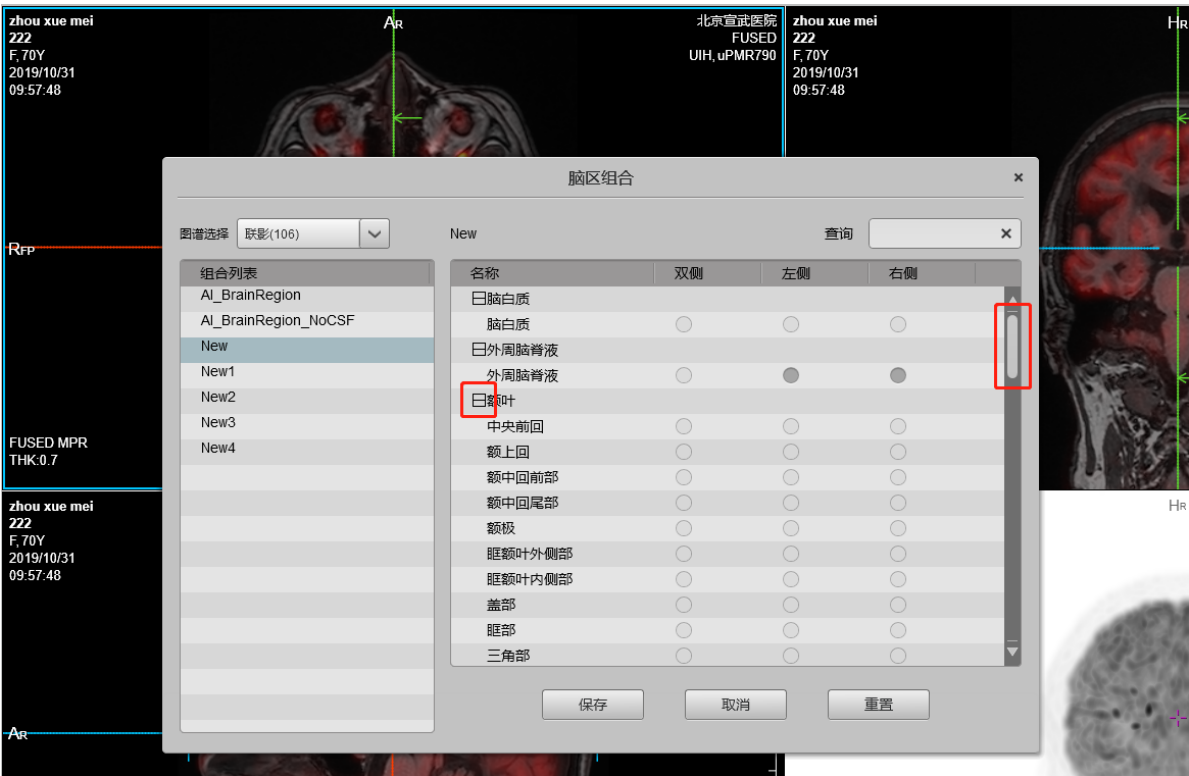
08-13~09-13

## 过程

ID	具体技术问题	TL
1	脑区编辑滚动条长短不一致DIM	张洋
2	为什么使用ObservableCollection而不用List	张洋
3	事件的创建销毁管理	马关坡
4	snoop的使用技巧	叶极宁

### 1. 脑区编辑滚动条长短不一致DIM

首先介绍一下脑区编辑滚动条长短不一致DIM



当折叠选项框中的选项后，使用滚动条向下滚动页面时，滚动条的长度会出现长短不一致的变化。观察折叠一个二级选项、折叠多个二级选项和不折叠二级选项时滚动条长短的变化情况以及分析设计的逻辑，可以猜测负责设置滚动条长短的属性是和总的选项框集合的 `count` 绑定的，因此当折叠或者打开二级选项时，集合的 `count` 会对应变小和变大，滚动条的长度也应该对应的变大和变小。

通过阅读代码，发现折叠二级选项时，其实是对被折叠的选项进行了隐藏处理，实际上存储所有选项的 `Collection` 里面的数据并没有改变。这样就造成了一个冲突，为了和用户看到的变化后的选项数对应，滑块的长短是应该要变化的，但是滑块的长短是和选项 `collection` 的 `count` 绑定的，而这个 `count` 并没有变化，所以就会造成滑动时出现滑块长度的跳跃性变化。

首先想到的解决DIM的思路是将折叠操作后用户可以看到选项数的 `new_count` 和滑块长度属性绑定，但是和TL交流后得知这一部分代码是在MCSF的common部分实现的，没有办法去修改。那么接着想到的是当折叠选项时把存储有所有选项的 `Collection` 中对应被折叠的 `item` 删除，那么和 `Collection` 的 `count` 绑定着的滑块长度也就能如预期的正常改变长度，当取消折叠再次打开二级菜单时又将之前删除的 `item` 添加回 `Collection`。至此解决问题的思路就清晰了，复制一个和原 `Collection` 初始数据完全一样的新 `Collection`，用来存储初始数据，当需要将删除的 `item` 添加回 `Collection` 时用来找到添加的 `item` 和添加的位置 `index`，剩下的需要注意的是算法方面，如何循环遍历两个 `collection`，如何找到插入的 `index` 等等。

---

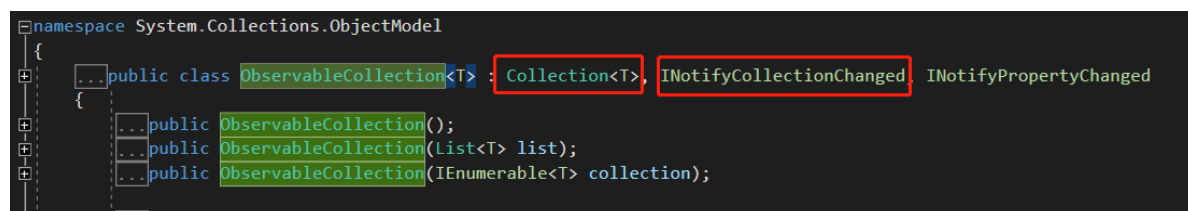
## 2. 为什么使用ObservableCollection而不用List

这个问题是在解决脑区编辑滚动条长短不一致DIM的时候延伸出的问题。如 1 中所述，通过增删 `collection` 中的 `item` 来动态改变滚动条长度，同样的选项列表的UI和 `collection` 也应该是对应改变的关系。

代码中使用的 `ObservableCollection` 继承自 `Collection`，尝试将 `ObservableCollection` 修改为 `List` 类型，编译运行发现折叠和打开二级选项时，多次操作后，选项列表的UI出现了混乱，显示的数据明显不正确。

通过分析UI显示的逻辑，首先推测可能是 `List` 和 `Collection` 在增删操作时有不同的特性导致的，一番搜索查询后发现两者的增删操作在此处的代码中应该是可以等效的，思路断掉了。TL提示这里的问题是关于绑定机制的，由此联想到之前学习绑定的使用时，自己写Demo也遇到过调试进去发现数据正常，但是UI显示不正常的问题，是因为没有发送消息告诉UI更新数据。

进代码看类的继承关系，果然 `ObservableCollection` 继承自 `Collection` 并实现了 `INotifyCollectionChanged` 接口中的相关函数。



```
namespace System.Collections.ObjectModel
{
    ... public class ObservableCollection<T> : Collection<T>, INotifyCollectionChanged, INotifyPropertyChanged
    {
        ... public ObservableCollection();
        ... public ObservableCollection(List<T> list);
        ... public ObservableCollection(IEnumerable<T> collection);
        ... public void NotifyCollectionChangedFromThread(CollectionChangedEventArgs e);
    }
}
```

当 `ObservableCollection` 实例中的数据变化时，通过 `RaisePropertyChanged("Property")` 来通知UI更新数据。显然这是 `List` 做不到的。

---

## 3. 事件的创建销毁管理

这是一个绘制ROI删除操作时的DIM相关问题。绘制多个ROI，然后删除，再重新绘制多个ROI，多次右键菜单分别点击隐藏ROI，会出现ROI无法隐藏的现象。

TL给出了问题分析的思路，经过一些尝试后，例如通过 `getHashCode()` 打印出当前ROI实例的 `hash code` 值发现这个值和目前已有的几个ROI值不一样，因此推断这个ROI实例不在我们的管控中，不受管控的ROI影像了正常绘制出的ROI的隐藏事件。由此猜测可能是之前创建又删除的ROI并没有完全销毁。通过查看代码发现，虽然之前创建ROI的实例虽然删除了，但是之前创建的ROI的隐藏事件没有进行 `--` 操作，这些不受管控的事件影响了后来创建的ROI正常的隐藏操作。

#### 4. snoop的使用技巧

通过学习snoop的使用，运用snoop来从DIM的UI显示找到关联的代码位置。WPF的UI分层叠加，开发人员有时候很难通过2D的视角来发现一些遮挡覆盖类的UI问题，通过snoop能够从UI的3D视角来发现这类问题。

snoop也能够将控件的树形结构展开，方便一些UI嵌套问题中，找到代码和UI的结构组成。

另外TL也介绍了snoop的实现原理，如何从一个进程去获得另一个进程的东西。这方面的知识在自动化测试中也可以用到。

---

#### 结果与心得

ID	具体技术问题	收获
1	脑区编辑滚动条长短不一致DIM	加深了对绑定的理解
2	为什么使用ObservableCollection而不用List	多F12进代码去看，很多问题会豁然开朗
3	事件的创建销毁管理	C#中的事件需要手动管理，这是开发中易踩的坑
4	snoop的使用技巧	了解工具原理和学习工具使用，提升效率

### 09-13~10-13

---

#### 过程

ID	具体技术问题	TL
1	应用的前后端通信方法	张洋
2		
3		
4		

---

#### 1. 前后端通信方法

这是在首次做新功能方面的工作时涉及到的知识点，具体功能是：`Design_Func_MR_Brain Analysis_`导出分割结果。需要实现的是前端UI添加一个 `Button`，点击 `Button` 后将脑分析 `mask` 数据以字节流的方式存储进磁盘。

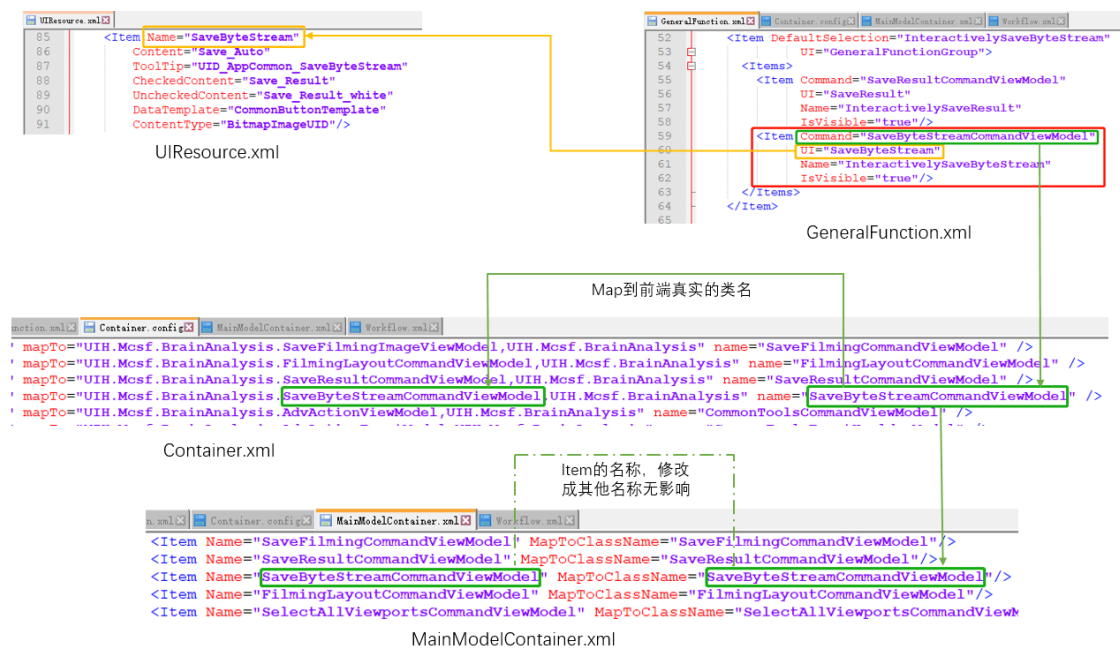
前端增加UI的方法不是这个问题的重点，因此简略描述一下。通过修改对应前端的 `FE` 文件夹下与需求相关的UI部分的配置文件，添加新的 `Button` 和 `Button` 的样式配置来增加新的UI。

```

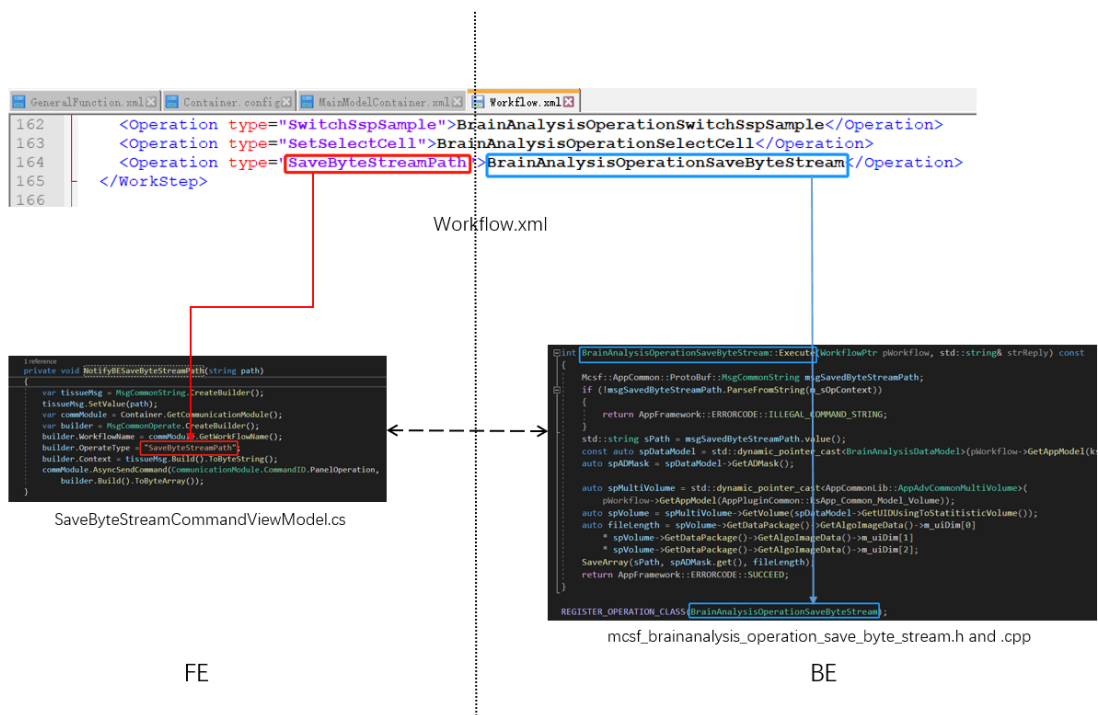
<Item DefaultSelection="InteractivelySaveResult"
      UI="GeneralFunctionGroup">
  <Items>
    <Item Command="SaveResultCommandViewModel"
          UI="SaveResult"
          Name="InteractivelySaveResult"
          IsVisible="true"/>
    <Item Command="SaveByteStreamCommandViewModel"
          UI="SaveByteStream"
          Name="InteractivelySaveByteStream"
          IsVisible="true"/>
  </Items>
</Item>

```

前端配置文件加载关系如图：



前后端通过 operation 通信方式：



## 结果与心得

ID	具体技术问题	收获
1	应用的前后端通信方法	
2		
3		
4		