# Undo,Redo功能设计文档说明书

## 功能需求

Canvas中元素的各种操作支持有限步骤的撤销、重做功能。具体操作暂定包括：单元素重命名，单元素删除，单元素添加，单元素移动，单条线删除，单条画线。**多元素的操作后续再考虑，目前看元素和关系的添加和删除必须考虑，不然操作不连贯会出问题。例如，如果多元素删除不考虑的话，当一个重命名操作入栈后，删除对应元素，则Undo操作就失去目标了。**

## 设计方案

### 2.1 数据结构

1，Undo,Redo两个固定大小的栈，目前默认大小各为100，既支持100步撤销、重做，已经能满足大部分需求，无限步骤的撤销、重做业务上没啥意义，还浪费内存。

2，ICommand接口，内含Undo, Redo两个动作，用来封装所有的单元素执行和回撤动作，每次执行后，要把对应文档置为Dirty。

3，UndoRedoManager,全局单例，用来管理Undo、Redo两个固定栈和协同界面Undo、Redo按钮事件等操作。其中每个文档设置Undo,Redo固定栈各一个，以ModelDocument为Key,{Undo，Redo}为Value,用Dict存储。当文档切换时，根据CurrentDocument切换到对应的固定栈,并重置界面按钮状态。

### 2.2 算法设计

1， 所有单元素完成操作的程序点，将相关上下文数据封装到ICommand中，然后Push进入Undo栈（由于是固定栈，所以当元素个数大于固定值时(默认100)，自动替换栈底元素），同时清空Redo栈。

2， 当点击Undo按钮时，Pop出Undo栈栈顶ICommand元素，然后执行Undo接口，回归对应操作，最后将ICommand元素Push进入Redo栈。

3， 当点击Redo按钮时，Pop出Redo栈栈顶ICommand元素然后执行Redo接口，执行对应操作，最后将ICommand元素Push进入Undo栈。

该操作与第二步对称。

1. 当Undo栈由空变不空时，Undo按钮Enable;当Undo栈由不空变空时，Undo按钮Disable;当Redo栈由空变不空时，Redo按钮Enable;当Redo栈由不空变空时，Redo按钮Disable;以上四种操作，由UndoRedoEvent的四种事件通知主界面
2. 在单元素的6种操作完成点，根据不同的操作类型封装ICommand接口所需的各种信息：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 操作 | 元素类型 | 进度 | 编码人 | 备注 |
| ElementRenameCommand | MoveDsElement | 完成 | 卢琪 |  |
| MoveOpElement | 完成 | 卢琪 |
| MoveRsElement | 完成 | 卢琪、夏天 |
| ElementAddCommand | MoveDsElement | 完成 | 卢琪、夏天 |  |
| MoveOpElement | 完成 | 卢琪、夏天 |
| MoveRsElement | 完成 | 夏天 | 点击配置完成之后会添加RsElement与关系 |
| ElementDeleteCommand | MoveDsElement | 完成 | 卢琪、夏天 |  |
| MoveOpElement | 完成 | 卢琪、夏天 |
| RelationAddCommand | ModelRelation | 完成 | 夏天 |  |
| RelationDeleteCommand | ModelRelation | 完成 | 夏天 |  |
| ElementMoveCommand | MoveDsElement | 完成 | 卢琪 | 编辑状态下的单元素移动操作。 |
| MoveOpElement | 完成 | 卢琪 |
| MoveRsElement | 完成 | 卢琪 |

6, 具体的Command类的构造函数，Undo,Redo执行逻辑依赖与具体的业务，同时要谨慎选择Command类的插入点。

### 2.3 代码实现

代码位于命名空间Citta\_T1.Core.UndoRedo下，

FixedCommandStack.cs, UndoRedo固定栈,双链表实现

ICommand.cs，命令类接口口, 约定Undo撤销，Redo重做

UndoRedoEvent.cs, 通知主界面撤销，重做按钮状态的事件

UndoRedoManager.cs, 统筹管理Undo,Redo行为

Command目录，实现各种具体命令类的业务逻辑

其他各种操作的代码插入点，具体操作具体分析。

### 2.4 逻辑描述

#### 单个算子的删除

算子的删除UndoRedo目的是在用户做出**显式**删除算子操作时，完成UndoRedo，即用户自己对算子进行删除时，完成UndoRedo。对于其他包含**隐式**删除算子的操作（删除Op时，会自动删除与之相连的Rs算子），Rs的删除与添加不予独立压栈

算子包括三种算子，MoveDtControl（以下简称Dt）、MoveOpControl（以下简称Op）、MoveRsControl（以下简称Rs），算子的删除情况如下

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **控件个数** | **控件类型** | **删除对象** | **删除方法** | **Undo** | **Redo** |
| 1 | Dt | Dt | 删自己 | 恢复自己 | 删自己 |
| 1 | Op | Op | 删自己 | 恢复自己 | 删自己 |
| 2 | Dt-Op | Dt | 删Dt和线 | 恢复Dt和线 | 删Dt和线 |
| 2 | Dt-Op | Op | 删Op和线 | 恢复Op和线 | 删Op和线 |
| 2 | Op-Rs | Op | 删Op和Rs和线 | 恢复Op和Rs和线 | 删Op和Rs和线 |
| 3 | Dt-Op-Rs | Dt | 删Dt和线Dt-Op | 恢复Dt和线Dt-Op | 删Dt和线Dt-Op |
| 3 | Dt-Op-Rs | Op | 删Op和Rs和线Dt-Op Op-Rs | 恢复Op和Rs和线Dt-Op Op-Rs | 删Op和Rs和线Dt-Op Op-Rs |
| 3+ | Dt-Op-Rs-Op | Dt | 删Dt和线Dt-Op | 恢复Dt和线Dt-Op | 删Dt和线Dt-Op |
| 3+ | Dt-Op-Rs-Op | Op | 删Op和Rs和线Dt-Op Op-Rs Rs-Op | 恢复Op和Rs和线Dt-Op Op-Rs Rs-Op | 删Op和Rs和线Dt-Op Op-Rs Rs-Op |

插入点选择如下

|  |  |  |
| --- | --- | --- |
| 动作类型 | 操作对象 | 插入点位置 |
| 添加 | Dt、Op | MainForm.NewDocumentOperator |
| 添加 | Rs | MovRsControlFactory.CreateNewMoveRsControl |

#### 单个算子的添加

算子的添加情况的UndoRedo目的是在用户做出显式添加算子操作时，完成UndoRedo。

算子包括三种算子，MoveDtControl（以下简称Dt）、MoveOpControl（以下简称Op）、MoveRsControl（以下简称Rs），算子的添加情况如下

|  |  |  |  |
| --- | --- | --- | --- |
| 动作描述 | 添加方法 | Undo | Redo |
| 添加Dt | 新增Dt | 删除Dt | 还原Dt |
| 添加Op | 新增Op | 删除Op | 还原Op |
| 添加Rs | 新增RS与线，更新Op状态 | 删除Rs与线，还原Op状态 | 还原RS与线，还原Op配置，还原Op状态 |

**Op算子的字典是readonly的，怎么办？**

#### 单个关系的添加与删除

关系的添加与删除UndoRedo目的是在用户做出**显式**添加与删除线操作时，完成UndoRedo，即用户自己对线进行“点击绘制”、“点击删除”时，完成UndoRedo。对于其他包含**隐式**添加与删除关系的操作（如配置Rs时，会自动生成关系，删除Rs时，会自动删除关系），关系的删除与添加不予独立压栈

|  |  |  |
| --- | --- | --- |
| 动作类型 | 插入点位置 | 备注 |
| 添加线 | CanvasPanel.CanvasPanel\_MouseUp |  |
| 删除线 | CanvasPanel.DeleteSelectedLinesByIndex | 不可以将插入点设置在CanvasPanel. DeleteRelation中，因为删除Op的时候会顺带删除Op和Rs之间的连线，这个时候只希望压一次栈 |

#### 多个算子的删除

多个算子的删除UndoRedo旨在当用户做出“批量删除”操作时，UndoRedo批量删除操作。区别于之前的单个算子、单个关系的UndoRedo，多个算子的删除只压一次栈，不重复调用单个算子的UndoRedo接口。

|  |  |  |
| --- | --- | --- |
| 动作描述 | Undo | Redo |
| 批量删除 | 删除多个算子与若干条相连的线 | 恢复多个算子与若干条相连的线 |

#### 多个算子的移动

多个算子移动的UndoRedo旨在当用户做出“框选移动”、“一键排版”等批量移动算子的操作时，完成多个算子移动的UndoRedo，整个UndoRedo过程只压一次栈，弹出一次栈。

|  |  |  |
| --- | --- | --- |
| 动作描述 | Undo | Redo |
| 批量移动 | 将多个算子位置重置为移动以前之前的位置，如移动期间世界坐标系发生改变，则世界坐标系的原点也需要重置 | 将多个算子的位置重置为移动后的位置，如移动期间世界坐标系发生改变，则世界坐标系的原点也需要重置 |

需要注意的是，世界坐标系原点+世界坐标，这两者的组合确定了一对世界坐标位置。

在一键排版等需要改变世界坐标原点的操作中，需要注意世界坐标原点的恢复。以一键排版为例

* + - 1. 点击一键排版时，首先需要保存世界坐标原点，然后保存各控件的世界坐标
      2. 做Undo的时候，涉及到保存当前坐标和还原之前坐标的问题。
         1. 保存当前世界坐标原点 oldWorldMap
         2. 将当前世界坐标原点重置为ICommand中的世界坐标原点， curWorldMap
         3. 使用curWorldMap将ICommand中的控件世界坐标转换为屏幕坐标并赋值
         4. 使用oldWorldMap将当前控件的屏幕坐标转为世界坐标后保存
      3. 做Redo的时候，同样涉及到保存当前坐标和还原之前坐标的问题，与Undo同理