# 思维导图

1.需求分析

思维导图又叫心智图，是表达发散性思维的有效的图形思维工具 ，它简单却又极其有效，是一种革命性的思维工具。思维导图把各级主题的关系用相互隶属与相关的层级图表现出来，把主题关键词与颜色等建立记忆链接。思维导图充分运用左右脑的机能，利用记忆、阅读、思维的规律，协助人们在科学与艺术、逻辑与想象之间平衡发展，从而开启人类大脑的无限潜能。思维导图因此具有人类思维的强大功能。

思维导图软件的功能在于通过专属的快捷键快速制作思维导图。其相比Photoshop和Illustrator等专业制图软件的优势在于能够快速创建制作美观又有序的思维导图，能够避免在制作过程中的思维隔断，做到图脑一体，所写即所想，而且还能够易于插入修改，重组排列。

我们针对思维导图的特点加入了颜色主题修改、节点样式修改、字体修改、连接线修改等等很实用的内容，方便用户在同一张思维导图中做到很大区分度的不同图像，方便发散思维。

## 2. 关键设计思路或方法

用QGraphicsView显示思维导图，用data()属性将 QGraphicsTextItem与QTreeWidget下的QTreeWidgetItem 联系起来，实现对树的操作。QGraphicsTextItem设置QGraphicsRectItem的子节点，接收鼠标选择事件，以及处理颜色、线形修改。

## 3. 详细设计

**Class介绍**

QTreeWidget Class: 用预先设置好的树形结构提供树状视图。

QTreeWidgetItem Class:为QTreeWidget提供可使用的物体。

QGraphicsView Class:提供了一个可以展示 QGraphicsScene内容的部件。

QGraphicsScene Class：提供了一个可以操作大量2D物体的平台。

QGraphicsItem Cass:是QGraphicsScene所有图形物体的基础。

QGraphicsRectItem class :提供了一个矩形物体，你可以把它加到QGraphicsScene.

QGraphicsLineItem class :提供了一条线，你可以把它加到QGraphicsScene.

QGraphicsTextItem class:提供了文本物体，你可以把它加到 QGraphicsScene来呈现一定格式的文本。

QPen Class:定义了一个QPainter应该怎样绘制一个图形的线条和轮廓。

QBrush Class:定义了被QPainter画出物体的填充。

QMenu class：提供了各种形式的主菜单。

QRadioButton widget:提供了一个带有 text label的 radio button。

QSpinBox Class:提供了一个spin box表面。

QMessageBox Class:提供了一个模型对话框，提供信息或者要求使用者回答问题，并且接受答案。

**基础事件**

**选择操作：**

将scene的selectionChanged()信号和QTreeWidget的setCurrentItem()联系起来，每次选择scene的某个节点的时候自动更新QTreeWidget的CurrentItem。

对QGraphicsTextItem和QTreeWidgetItem进行setData()操作，对应的节点Data相同。

初始时设置selection和currentItem。

**树的绘制：**

后序遍历，每个叶节点为一层，父节点的位置为子节点层数的平均值（最大+最小）/2。

**线和背景的绘制**

背景QGraphicsRectItem设置为QGraphicsTextItem的child，位置根据QGraphicsTextItem确定（即相对坐标(0,0)）

根据不同的层使用不同的brush。共五层，循环。

线条设置一个parent，以实现对QGraphicsLineItem的操作。位置确定和QGraphicsTextItem类似，在绘制y方向线条时要考虑到最大子节点和最小子节点的位置。

根据不同层使用不同的pen。共五层，循环。对第一层需要特殊处理。

**插入删除操作：**

其中插入操作为插入该节点的子节点，删除操作为删除此节点以及其下的所有子节点。

用设置parent的形式存储所有的QGraphicsTextItem，结合QTreeWidget对所有的QGraphicsTextItem依次进行搜索。

考虑到插入的节点一定为叶节点并且在选中的节点前，对后序遍历第一次读取到的位置进行新建QGraphicsTextItem操作。

最后重新对Data进行设置。

**外观事件**

**颜色修改**

接收对应的clicked()事件，并根据对应信号进行处理。

由于无法找到非常合适的用于重绘QGraphicsRectItem和QGraphicsLineItem的命令，在每次触发事件时(修改颜色，创建节点等)将已有QGraphicsRectItem和QGraphicsLineItem移除重新创建。

其中主题包括背景和边框线。

**线型修改**

接收对应的clicked()事件，并根据对应信号进行处理。

将已有QGraphicsRectItem和QGraphicsLineItem移除重新创建。

采用Qt内置的五种线型。

**线宽修改**

接收valueChanged(int)信号。

分别存储线条线宽和矩形框线宽。

在每次绘制前根据绘制的是线条还是矩形框重新setWidth()，即定义线宽。

**字体修改**

接收fontComboBox的currentFontChanged(QFont)信号，并且设置QFont为scene的Front。

遍历QGraphicsTextItem，用setFont(scene.font())更新字体。

**其他事件**

**帮助**

使用QMessageBox提供帮助事件。

**关于**

使用QMessageBox提供帮助事件。

## 4. 开发体会&小结

感觉走了不少弯路，Qt在网上的很详细的文档比较少……然后就慢慢学会了看帮助文档，为了实现这个功能……可以算是挺狼狈的。

大概也说一下小组的讨论和方向吧。先是看资料了解QT的各种类，想了很多的功能，最后选择用QGraphicsView来显示，其实可能单纯因为有QGraphicsTextItem这个类，觉得和我们的思路蛮符合的，而且QGraphicsView可以有滚动条的话应该可以比较好地呈现我们思维导图的想法。

在中期的时候看了一些比较早的Qt的教程，因为好像缺少很新的系统的教程，然后就先把主界面希望实现的写了，在中期报告里可以看到吧。很多资料说要是自己写代码的生成界面的话自由度比较高……然而还是觉得Designer会方便很多hh。

对树的构建是比较困扰的一个部分，一开始没有意识到QGraphicsItem可以设置父节点和子节点，如果下一次要再次实现的话会直接设置QGraphicsItem的父子结点的。虽然在DS里有比较详细的学到树但是很难直接应用，如果考虑到应用的话要存四个结点分别访问子节点父节点上一个同级下一个同级……要考虑增删……还要存每个点的TEXT……

因为希望Qt有内置的类可以处理这个功能，所以在Designer里找的时候对Tree这种关键词特别敏感……然后就考虑用QTreeWidget，然后迎来了一堆bug……

后来写QTreeWeidget，一开始非常顺利，插入删除这一部分还好，网上有给出比较详细的代码，一开始的思路是根据QTreeWeidget直接生成QGraphicsView的，一开始也还顺利。就是每次编辑操作后重新生成QGraphicsView。这里主要就是，QGraphicsView和QGraphicsScene的关系稍微折腾了一下，还有QGraphicsTextItem的计算。

每一次增删操作都全部删除重新生成，相当于重新绘图。但问题是QTreeWeidget的QTreeWidgetItem文本修改找不到对应的信号，只能在有别的信号触发的情况下才能完成更新。

后来生成了之后发现QGraphicsView的QGraphicsTextItem的有编辑文字的功能，然后希望能够直接把QGraphicsTextItem的文本更新到QTreeWidgetItem上而不是通过QTreeWidgetItem更新QGraphicsTextItem。

然而QGraphicsTextItem依然找不到文本修改的信号……甚至连读取文本内容的函数都没找到……

后面差不多重新再来……考虑通过移动QGraphicsTextItem而不是每次重新生成来实现。这样可以保留每次在QGraphicsTextItem上的文本。

然而，想了好久的办法把QTreeWidget的QTreeWidgetItem和QGraphicsTextItem联系起来，可以在选择QGraphicsTextItem的时候选中对应的QTreeWidgetItem。

QTreeWidget有提供先序遍历的内置函数，可我的整个绘图建立在后序遍历的基础上orz，感觉这个类不是这样用的……

然后就把QGraphicsTextItem全部设成某个结点的child，遍历。给每一个QGraphicsTextItem setData然后根据Data判断是哪个结点……算法复杂度咳咳

顺便遍历的时候QGraphicsItem没有办法直接返回……本来想返回某个Data的Item 结果返回不了……最后是返回在那个child里的序号……

这一块溢出了好几次……感觉这一块很容易出问题……

反正总算是弄好了。

因为一开始一直是生成文本一直为同一个QString 所以感觉很顺利，长得很像一棵树。

然后为了QGraphicsTextItem能够链接到QTreeWidgetItem，于是给QTreeWidgetItem也set了data……

发现QGraphicsTextItem的select事件终于有信号了……

然而可以编辑的文本，点一下基本就变成了编辑文字而不是释放select信号。于是我给QGraphicsTextItem加了个底…就是QGraphicsRectItem。顺便处理了一下后面的加颜色什么的事件。

然后想稍微录一下报告下进程

然后从这里开始报警顺便进入困了最久的地方

莫名其妙插入插入弄着弄着所有的结点顺序都错了？？

是直接顺序错掉的那种……根本没有规律……

最后发现是QTreeWidget 返回Child的顺序……好像和Child的Child数量有关……

帮助文档只写了以一定顺序返回……谁想到是这个顺序……

可谁知道问题出在这呢……

QTreeWidgeItem的几个结点调用的顺序，想不到好的解决方法，上了个数组直接Qsort……结果每层的数量都有一定限制……

然后关于Tree和TextItem该啥时候setData……要是设置的太早了就和本来那个位置的Data冲突了……

反正这一块调了超级久……

我只能说QTreeWidget不是这么用的……明明都是树啊orz

然后加删除……因为要删掉整棵子树，而不是删掉对应点就好了……只能说还是很绕……

反正最后终于弄好了……感觉看了好几天漫天乱跑的各种奇怪的节点……自己的名字在程序上显示那一刻真的蛮开心的~

颜色和线的话，主要在之前的基础上写。考虑把背景设置成文字的Child，这样子添加删除查找都方便。结果没有考虑到Child是相对坐标，结果底图的位置一直很奇怪……

加线条的话一开始感觉差不多，就是y方向的线条要记录比较多的位置，然后加完我发现我可以移动文本节点但我怎么移线条呢……

于是我加了个parent把所有线条设置成child……每次都删光重建……

因为程序不大所以可以稍微少考虑点性能……

还有不知为何没找到修改背景矩形的颜色和边框的函数。结果后来找啊找……又变成了每次删光重建……

杨靖岚：

信号和槽，开始看的时候觉得（这有啥难的），然后就是无数次的报错。

主要是，找不到信号啦，找不到槽啦（也就这两种错了吧）……

一开始还没看到调试的时候VS的框会显示，调试全靠断点+猜。还有比如说槽不能带参数，想象信号相当于调用某个槽函数，但好像机制不太一样；比如说没加到 public SLOT里面去。可能只有真的自己去实践才能知道这些东西。

Designer的布局也没想象中简单，一开始是采用悬停的工具栏，后来效果不好改回了原来的样子，为了让它适应放大缩小和隐藏工具栏也弄了蛮久。对Designer的很多功能还不熟练有待摸索。

在思维导图的节点类的设计时，一开始是自己设计了一个节点类，虽然对于功能的实现比较便捷和直观，但由于所需要的指针数目比较多，感觉不是很实用。后来发现了qt自己treewidget,这个qt自带的树的类很好的实现了关于树的操作，并且可以很方便调整和获取当前节点，和键鼠的交互事件比较方便。但是由于我们不能对这个类进行修改，这就导致如何将treewidget和屏幕上的文本框联系在一起成为了开发的一个难点，最后使用了树的遍历来寻找当前节点，略微影响了一点程序的效率，不过还是解决了这个问题。

主菜单上用Qbutton加了一些事件，因为还是有些仓促一些复制粘贴以及保存的按钮有待实现。

感觉一个程序真的是需要所有人不断的努力，和大家合作的过程中收获了很多。