# 树形控件设计

设计树形控件时，最重要的是树形视图内所有展示控件的组织问题，主要是树形视图内用来包含展示控件的数据结构。

树形试图在显示其内部展示控件的过程中，包含以下操作

① 扩展，展开某个展示控件的子节点

② 收缩，收缩某个展示控件的子节点

树形视图内部的数据结构应该能方便简单的支持上述两种操作，目前考虑的方法是顺序结构，树形结构。

## 数据结构设计

### 顺序结构（插入，删除慢，搜索快）

若树形控件内部展示控件的数据结构使用顺序结构，则在初始化过程中，需要将所有展示控件包括父控件及其全部子控件都加载到顺序结构数组中，为每个控件设置一个扩展标记E，表示该控件当前的状态对应展开还是收缩。

下面分别针对扩展和收缩进行模拟操作来验证该结构的优点和缺点。

### 多级目录结构（插入，删除快，搜索慢）

将所有顶层节点放在同一层，每个顶层节点的子节点由对应顶层节点进行索引，形成多级目录结构。

针对一个共有n层，每层有m个节点的测试数据树而言:

1. 初始化

从第一个顶层节点开始，首先确定该节点的绝对位置偏移，随后确定其所有展开子节点的绝对位置偏移，之后的顶层节点重复此操作。

显示的时候根据当前显示的第一个控件的坐标，及当前树形控件能展示的最多的控件数量，从顶层节点及其所有子节点一直搜寻，直到达到显示目标。

1. 展开

节点展开的时候，向树形控件发送展开信号，携带展开节点的绝对位置偏移参数。

树形控件使用该节点的绝对位置偏移参数从第一层节点进行二分查找，时间复杂度nlog（n\*m）。

找到对应节点后，展开该节点的所有子节点为其设置内容偏移，时间复杂度O（n\*m）。

## 动画设计

包括添加，删除，展开，收缩动画

### 已有数据结构

1. 节点数据结构

动画设计中用到的属性如下:

⑴ \_ch: 用于表示本节点的当前实际高度

⑵ \_h: 用于指定本节点的总高度

⑶ \_dh: 节点处于动画状态时的增量

⑷ \_state: 节点当前的动画状态

⑸ \_contentOffset: 节点在内容页相对0地址的位置偏移

(6) \_visibleLen: 本节点展开时，可见的子节点数目

(7) \_isExpand: 表示本节点是否处于展开状态，true代表的展开，false为收缩

② 树形控件数据结构

⑴ \_allItem: 以多级目录结构存储所有节点，相当于所有节点的逻辑视图，下文称内容页。

### 提供方法

1. 节点内部:
2. void addAnimationStart(int sv, int ev, int t)

sv: 为\_ch设置的起始值

ev: 为\_ch设置的最终值

t: 持续时间毫秒

流程:

1. 设置状态为Adding
2. 修改\_ch为0，\_h为ev
3. 根据t 计算\_dh
4. 启动定时器开始节点添加的显示动画

备注: 进入定时器事件后根据节点当前状态修改\_ch，触发窗口重绘，时间复杂度高

1. void expandAnimationStart(int t );

t: 同上

流程：

1. 修改状态为Expanding
2. 获取子节点高度，并设为\_cth属性值
3. 令\_cch为0，根据t计算得\_dh
4. 启动定时器开始节点扩展的动画
5. void collapseAnimationStart(int t)

t: 同上

流程:

1. 修改状态为Collapsing
2. 根据当前的\_cch和t计算得\_dh
3. 启动定时器开始节点收缩动画
4. int internalUpdateOffset(int off, int& rh, NavigateItem\* root)

off: 指定节点在树形控件内容页中和起始点0处的位置偏移。

rh: 用来表示本节点剩余的显示高度，为动画而添加

root: 函数内部使用的递归参数

流程:

1. 判断rh是否大于0，大于0进入2，小于等于0返回已访问的节点总高度h。
2. root->\_contentOffset = off
3. 比较rh和root->\_h, 若大于等于进入4，小于进入5
4. 令root->\_ch = root->\_h，rh = rh – root->\_h
5. 令root->\_ch = \_rh, rh = rh-root->\_h
6. h += root->\_ch,
7. 检查是否展开，若未展开进入9
8. 遍历所有子节点，重复递归操作，遍历条件需满足rh> 0，修改visibleLen
9. 返回访问的所有节点高度和，函数直接返回。

**⑶ void timerEvent(QTimerEvent\*);**

流程:

① \_state == Adding

1. 比较\_ch和\_h，若大于等于则进入3，若小于则进入2
2. 令\_ch = \_ch + \_dh, 启动下一次定时器，进入通用逻辑
3. 此时\_ch已经大于\_h, 动画已经完成，停止计时器，状态设Normal

② \_state == Expanding

1. 比较\_cch 和 \_cth, 若大于等于则进入3，若小于则进入2
2. 令\_cch = \_cch + \_dh, 启动下一次定时器，进入通用逻辑
3. 此时\_cch已经大于\_cth, 动画已经完成，停止计时器，状态设Normal

③ \_state == Collapsing

1. 比较\_cch是否大于0，若小于等于则进入3，若大于则进入2
2. 令\_cch = \_cch – dh, 启动下一次定时器，进入通用逻辑
3. 动画执行完成，停止计时器，状态设Normal

通用处理逻辑:

1. 发送signals void updateTree(NavigateItem\* item)信号令树形控件重绘

信号(回调函数):

1. signals void updateTree(NavigateItem\* item)

说明: 此信号对应树形控件的onUpdateTree，用于更新树形控件的显示内容。

1. 树形控件
2. void onUpdateTree(NavigateItem\* item);

item: 触发重绘的控件

流程

1. 以当前item为起始点调用updateContentOffset
2. **int updateContentOffset(NavigateItem\* item, int off)**

**item: 修改内容页内容的起始位置对应的节点**

**off: item在内容页对应的位置偏移**

**流程**

1. **函数从传入的item节点处开始，以off为起始值，修改从item往后(包括item)的所有可见节点的位置偏移，更新顺序为item，item的所有子节点，item的所有兄弟节点，item的父节点后面的所有节点。**
2. **触发窗口重绘**

备注: 时间复杂度较高，若共有n个节点，时间复杂度O(n)。

1. void updateView(NavigateItem\* st\_item, int d1)

说明:

1. 从内容页中以st\_item为起始点，d1为起始位置，收集一页控件用于显示。
2. 窗口的重绘函数之一

备注: 内容页现在要求保证实时对应动画的每一帧，视图的每一帧控件的坐标和控件集合都从内容页中挑选，不做任何其他处理。

### 添加动画

当节点被添加到树形控件时，节点对应的控件高度从0开始逐渐增加到指定的高度，同时在高度改变的过程中，树形控件的其他控件跟随该节点高度变化而变化。

添加操作分顶层节点和子节点，具体实现流程如下:

1. 添加顶层节点
2. 将该节点添加到内容页，设置节点\_h等属性
3. 调用void addAnimationStart(int sv, int ev, int t)方法
4. 添加子节点
5. 将该节点添加到内容页
6. 调用void addAnimationStart(int sv, int ev, int t)方法

### 展开动画

流程:

1. 当节点被展开时，设置节点的\_isExpand为true，否则无法被**int updateContentOffset(NavigateItem\* item, int off)函数识别到**
2. 调用void expandAnimationStart(int t );执行展开动画

### 收缩动画

流程:

1. 当节点被收缩时，设置节点的\_isExpand为false，否则无法被**int updateContentOffset(NavigateItem\* item, int off)函数识别到**
2. 调用void expandAnimationStart(int t );执行展开动画
3. 动画执行结束

### 删除动画

流程:

1. 收缩被删除节点
2. 启动删除动画
3. 动画结束移除节点

## 其他

### 增量更新算法

1. 引言

由于当前的内容视图更新机制会导致每次触发更新都要执行一次时间复杂度为O(n)的操作, n为节点总数数量级别，导致浪费了很多时间，若将每次更新都设置为一个固定值，每次触发更新都从引起更新的节点向后更新，固定更新高度，从而将更新操作简化为常数时间复杂度。但这也同时在程序中引发了一个问题，当触发更新操作的时候，每次只更新固定高度，若每次触发更新操作的都是同一个节点，更新将从同一个内容偏移开始，导致每次更新都有重复更新节点，做了很多的无效的操作，因此考虑使用增量更新操作。

1. 原理

在每次触发更新操作都是同一个节点的情况下，通过查询其上一次更新的最后一个节点，从该节点继续向下更新，由于触发更新操作是由于节点的高度或状态发生变化，因此每次更新操作中只有唯一一个高度和状态变化的节点，也就是引起此次更新的节点，因此只需要对该节点使用增量更新操作。在增量更新算法下，共需要触发常数次更新操作，每次更新操作都是常数时间复杂度，因此更新内容视图的整体时间复杂度仍为常数，每次更新操作对应一帧。