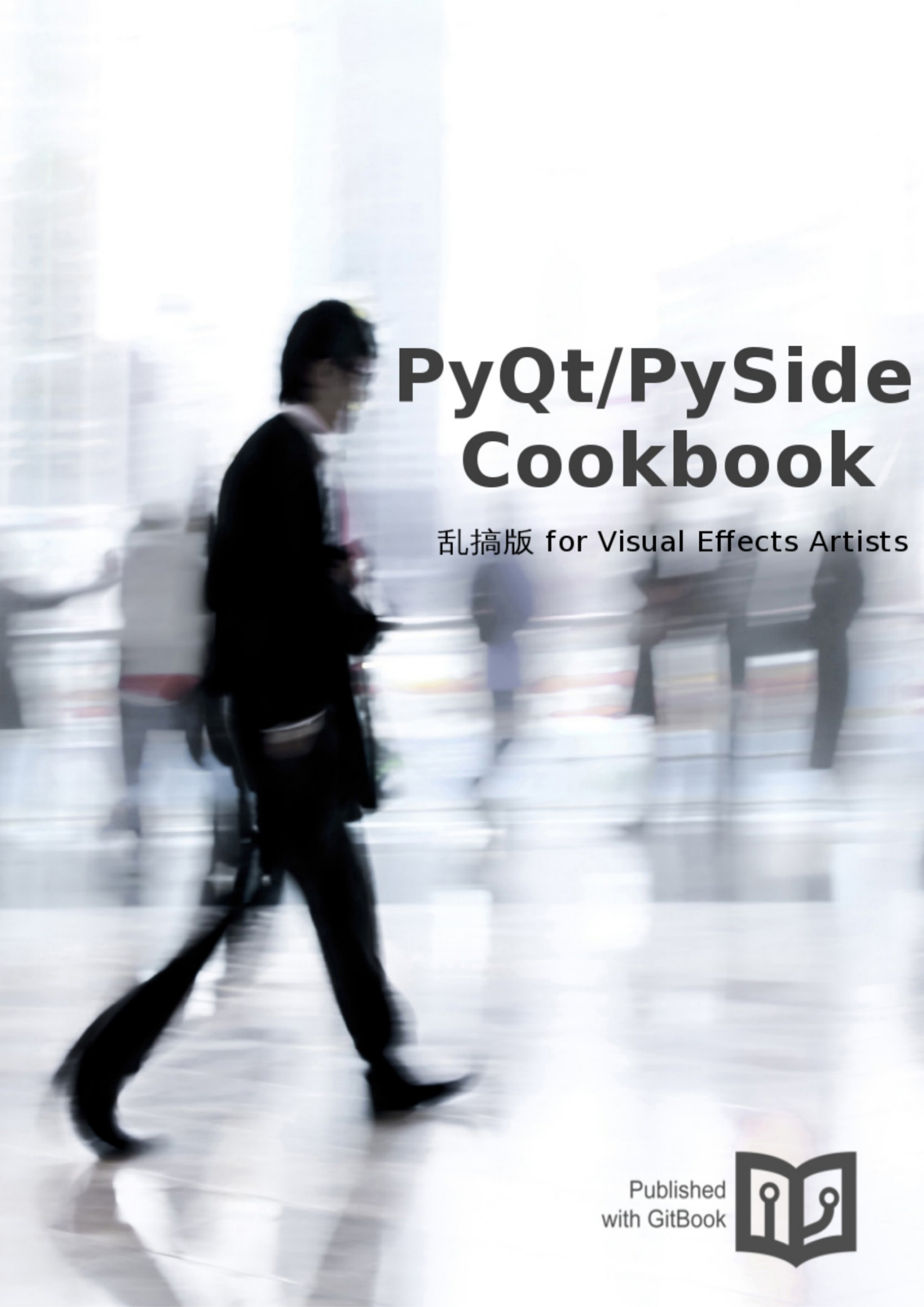


PyQt/PySide Cookbook

乱搞版 for Visual Effects Artists

Published with GitBook

Table of Contents

1. 简介
2. 资源和翻译
3. 信号和槽
    i. 动态创建一系列组件并绑定信号和槽
4. Widget
    i. 在Maya内部运行的模板
    ii. 带overflow效果的按钮
    iii. QAction
5. List Widget
    i. 遍历List Widget
    ii. 同时勾选多个items
6. Tree
    i. 半透明效果拖拽
    ii. 自定义drop indicator
    iii. 拖拽带itemWidget的treeWidgetItem
7. Table
8. 对话框
9. 打包发布
10. StyleSheet
11. Designer
    i. Promoted Widgets
12. Graphics View

PyQt/PySide Cookbook

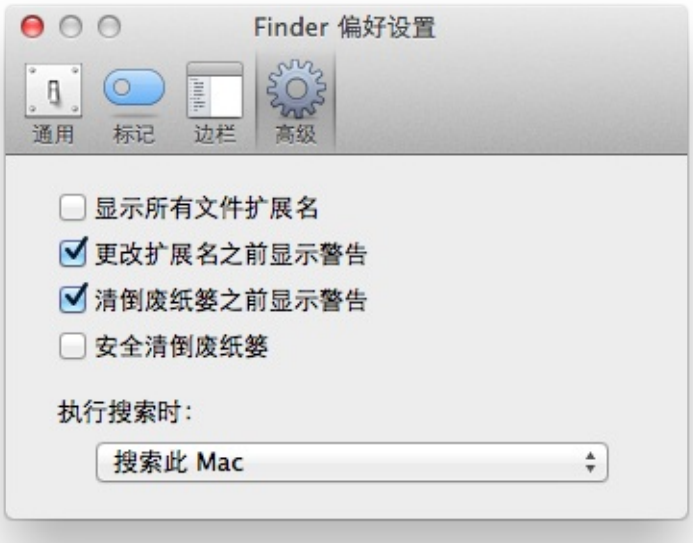
该项目主要用于收集PyQt/PySide开发中碰到的问题的解决方案，以及一些案例。

主要代码说话，解决问题为主，少讲或不讲原理。

资源和翻译

信号和槽

动态创建一系列组件并绑定信号和槽



如上图所示的需求，需要创建4个QCheckBox，一种做法是直接Designer中设计，这种做法适合知道选项的情况。还有种情况是这些选项是通过配置文件读出来的，或者是数据库中取出来，或者其他情况得到的，这时候就需要动态创建了。

解决思路是循环一个列表，创建对象，插入布局即可。这里稍微增加一些复杂性，对所创建的QCheckBox对象进行信号和槽的绑定，并处理信号，这时候我们可以通过 `sender()` 方法来获得是哪个对象发出的信号。

其他的控件类似，这里就用QCheckBox做例子了。

下面是完整代码(PyQt4/Python2.7)：

```
# -*- coding: utf-8 -*-
from PyQt4 import QtGui, QtCore

class Widget(QtGui.QWidget):
    def __init__(self, parent=None):
        QtGui.QWidget.__init__(self, parent)

        layout = QtGui.QVBoxLayout()

        items = [(0, 'Python'), (1, 'Golang'), (2, 'JavaScript'), (3, 'Ruby')]
        for id_, text in items:
            checkBox = QtGui.QCheckBox(text, self)
            checkBox.id_ = id_
            checkBox.stateChanged.connect(self.checkLanguage)
            layout.addWidget(checkBox)

        self.lMessage = QtGui.QLabel(self)
        layout.addWidget(self.lMessage)

        self.setLayout(layout)

    def checkLanguage(self, state):
        checkBox = self.sender()    # 获取发射信号的对象

        if state == QtCore.Qt.Unchecked:
            self.lMessage.setText(u'取消选择了{0}: {1}'.format(checkBox.id_, checkBox.text()))
        elif state == QtCore.Qt.Checked:
            self.lMessage.setText(u'选择了{0}: {1}'.format(checkBox.id_, checkBox.text()))


if __name__ == '__main__':
    import sys

    app = QtGui.QApplication(sys.argv)
    widget = Widget()
    widget.show()
    sys.exit(app.exec_())
```

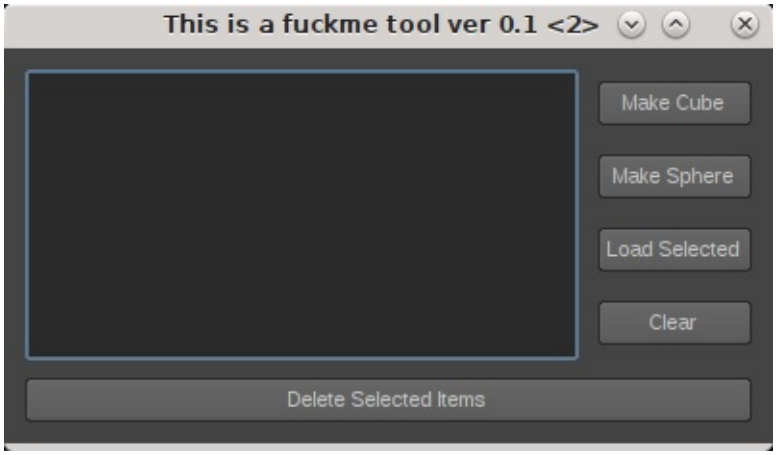
效果截图：

Widget

在Maya内部运行的模板

Note:

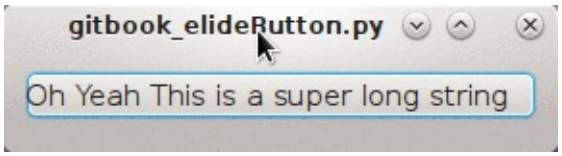
- 因为 `self.ui = uic.loadUi(uiFile, self)` 这一行把self传给了 `loadUi` function作为baseinstance,所以之后可以直接用 `self.myWidget` 来引用widget,而用不着 `self.ui.myWidget` 。
- 因为想再次按下shelf button时，关掉已有窗口，再开一个新 的，所以使用了全局变量win,这个变量显然应该对每个工具都要起成不同的名字。
- 如果在Maya里,qApp已经在global namespace里了。
- 模板抄袭自Nathan网站



```
import os
import sys
from PyQt4 import QtCore, QtGui
from PyQt4 import uic

uiFile = os.path.join(os.path.dirname(__file__), 'pyqt_example.ui')

try:
    import maya.OpenMayaUI as apiUI
    import sip

    def getMayaWindow():
        ptr = apiUI.MQtUtil.mainWindow()
        return sip.wrapinstance(long(ptr), QtCore.QObject)
except:
    pass


class MyWindow(QtGui.QDialog):

    def __init__(self, parent=None):
        super(MyWindow, self).__init__(parent)
        self.ui = uic.loadUi(uiFile, self)
        self.ui.show()


def main():
    if QtGui.qApp.applicationName().startsWith('Maya'):
        global win
        try:
            win.close()
        except:
            pass
        win = MyWindow(getMayaWindow())
    else:
        app = QtGui.QApplication(sys.argv)
        win = MyWindow()
        sys.exit(app.exec_())

if __name__ == "__main__":
        main()
```

带overflow效果的按钮

想实现的效果是，当QPushButton上的label过长的时候，自动把过长的部分...类似css里的overflow效果



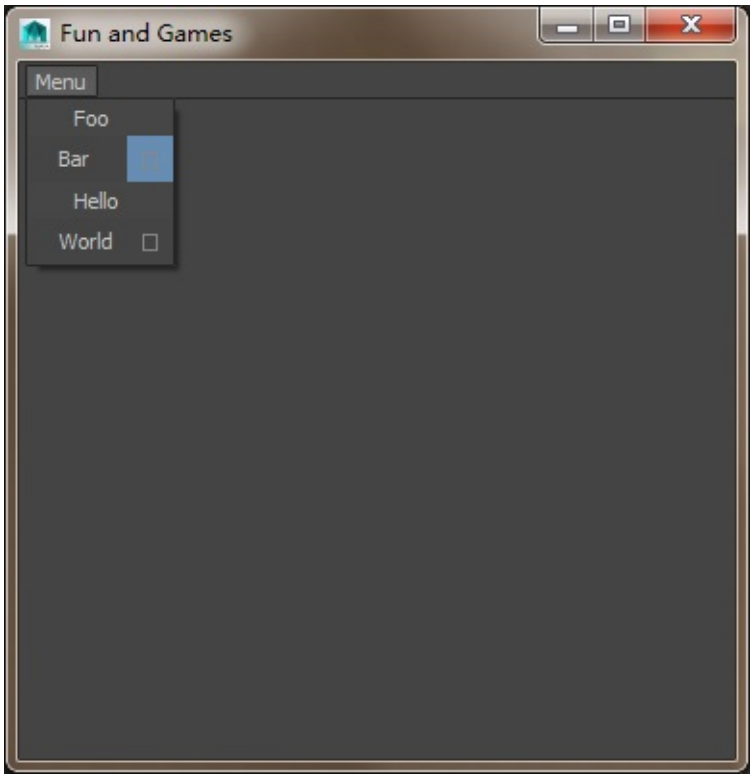
代码如下：

```
#!/usr/bin/env python2
import os
import sys
from PyQt4 import QtGui, QtCore
from PyQt4.QtCore import Qt, QString


class ElideButton(QtGui.QPushButton):

    def __init__(self, parent=None):

        super(ElideButton, self).__init__(parent)
        font = self.font()
        font.setPointSize(10)
        self.setFont(font)

    def paintEvent(self, event):
        painter = QtGui.QStylePainter(self)

        metrics = QtGui.QFontMetrics(self.font())
        elided = metrics.elidedText(self.text(), Qt.ElideRight, self.width())

        option = QtGui.QStyleOptionButton()
        self.initStyleOption(option)
        option.text = ''
        painter.drawControl(QtGui.QStyle.CE_PushButton, option)
        painter.drawText(self.rect(), Qt.AlignLeft | Qt.AlignVCenter, elided)


class TheUI(QtGui.QDialog):

    def __init__(self, args=None, parent=None):
        super(TheUI, self).__init__(parent)
        self.layout = QtGui.QVBoxLayout(self)
        self.button = ElideButton('Oh Yeah This is a super long string')
        self.layout.addWidget(self.button)
        self.setMinimumWidth(20)

if __name__ == '__main__':
    app = QtGui.QApplication(sys.argv)
    gui = TheUI()
    gui.show()
    app.exec_()
```

其实这个例子有个小缺陷，label左侧应该有些空隙（由style决定的）但是此处没有继承到。

3.1.QAction

类似maya里的下拉菜单




```
import os
import sys

from PyQt4 import QtGui, QtCore

MY_ICON = [
      '16 16 4 1',
      '  c None',
      '. c #c6c4c4',
      'l c #888888',
      ': c #909090',
      '                ',
      '                ',
      '                ',
      '                ',
      '   llllllll     ',
      '   l      l     ',
      '   l      l     ',
      '   l      l     ',
      '   l      l     ',
      '   l      l     ',
      '   l      l     ',
      '   l      l     ',
      '   llllllll     ',
      '                ',
      '                ',
      '                ',
      '                ',
      '                ',
]

## Custom Action used to provide an action and a clickable icon
```

```
#
class ExtendedQAction(QtGui.QWidgetAction):
	def __init__(self, label, mainAction, secondaryAction, *args, **kw):
		QtGui.QWidgetAction.__init__(self, *args, **kw)

		myWidget = QtGui.QWidget()
		myLayout = QtGui.QHBoxLayout()
		myLayout.setSpacing( 0 )
		myLayout.setContentsMargins( 0, 0, 0, 0 )
		myWidget.setLayout(myLayout)
		myLabel = ExtendedQLabel(label)
		myIcon = ExtendedQLabel()
		myIcon.setPixmap(QtGui.QPixmap(MY_ICON))
		myLayout.addWidget(myLabel, stretch=1)
		myLayout.addWidget(myIcon, stretch=0)

		## Hack in the hover colors to a style sheet
		#  The global stylesheet is not controlling the highlight color.
		#  It would be good to figure out how to avoid hardcoding styles here.
		defaultHLBackground = "#%02x%02x%02x" % myWidget.palette().highlight().color().getRgb(
		defaultHLText = "#%02x%02x%02x" % myWidget.palette().highlightedText().color().getRgb()|
		myLabel.setStyleSheet('padding-left:14px')
		myWidget.setStyleSheet("QWidget:hover { background:%s; color: %s;} QWidget { padding: 4p


		myIcon.setToolTip("Secondary Action Toolttip" )

		self.connect(myLabel, QtCore.SIGNAL('clicked()'), mainAction)
		self.connect(myIcon, QtCore.SIGNAL('clicked()'), secondaryAction)

		self.setDefaultWidget(myWidget)


## Clickable QLabel, this was the path of least resistance for making a
#  clickable image/label.
#
class ExtendedQLabel(QtGui.QLabel):

	def __init(self, parent):
		QtGui.QLabel.__init__(self, parent)

	def mouseReleaseEvent(self, ev):
		self.emit(QtCore.SIGNAL('clicked()'))


################################################################################
# Usage of the dual action menu items.
################################################################################
class MyMainWindow(QtGui.QMainWindow):
	def __init__(self, *args, **kwargs):
		QtGui.QMainWindow.__init__(self, *args, **kwargs)

		regularAction = QtGui.QAction('Foo', self)
		extendedAction = ExtendedQAction('Bar', self.mainAction, self.secondaryAction, self)
		regularAction2 = QtGui.QAction('Hello', self)
		extendedAction2 = ExtendedQAction('World', self.mainAction, self.secondaryAction, self)

		menubar = self.menuBar()
		myMenu = menubar.addMenu('&Menu')
		myMenu.addAction(regularAction)
		myMenu.addAction(extendedAction)
		myMenu.addAction(regularAction2)
		myMenu.addAction(extendedAction2)
```

```
            self.setWindowTitle('Fun and Games')

        def mainAction(self):
            print "performing main action"

        def secondaryAction(self):
            print "performing sectiondary action"

def main():
        app = QtGui.QApplication(sys.argv)
        funAndGames = MyMainWindow()
        funAndGames.show()
        sys.exit(app.exec_())

if __name__ == '__main__':
        main()
```

List Widget

遍历List Widget

最直接的方法如下：

```
items = []
for index in xrange(self.listWidget.count()):
    items.append(self.listWidget.item(index))
```

但是这样很不pythonic，基于一般越短的代码就是越正确的代码的原理，可以像下面这样：

```
all_items = self.listWidgetA.findItems(QString('*'), Qt.MatchWrap | Qt.MatchWildcard)
```

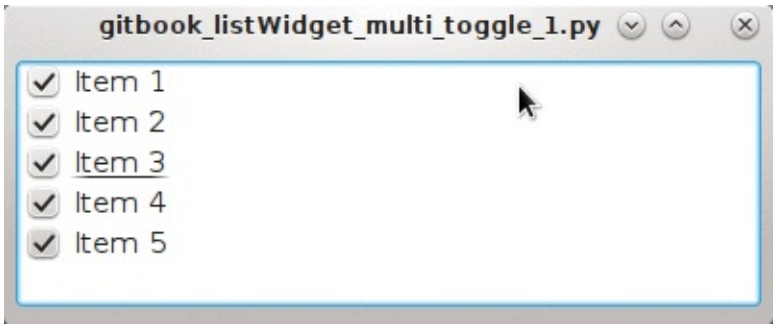
但是这相当于把所有的items全部存到一个list里面了，会费资源，还是不好。

可以考虑如下方式，在listWidget的subclass里定义一个iterAllItems method,其实他是一个generator。

```
def iterAllItems(self):
    for i in range(self.count()):
        yield self.item(i)
```

同时勾选多个items

框选多个item之后，用空格键可以勾选/去选多个item，效果如下图所示：



方法是reimplement keyPressEvent，只要按键是空格键，就勾选/去选选择了的items，代码如下

```python
```

from PyQt4 import QtGui, QtCore from PyQt4.QtCore import Qt, QString import sys import os

class ThumbListWidget(QtGui.QListWidget):

```
def __init__(self, type, parent=None):
    super(ThumbListWidget, self).__init__(parent)
    self.setIconSize(QtCore.QSize(124, 124))
    self.setSelectionMode(QtGui.QAbstractItemView.ExtendedSelection)
    self.setAcceptDrops(True)
    self.setSelectionRectVisible(True)

def keyPressEvent(self, event):

    if event.key() == Qt.Key_Space:
        if self.selectedItems():
            new_state = Qt.Unchecked if self.selectedItems()[0].checkState() else Qt.Checked
            for item in self.selectedItems():
                if item.flags() & Qt.ItemIsUserCheckable:
                    item.setCheckState(new_state)

        self.viewport().update()

    elif event.key() == Qt.Key_Delete:
        for item in self.selectedItems():
            self.takeItem(self.row(item))

def iterAllItems(self):
    for i in range(self.count()):
        yield self.item(i)
```

class Dialog(QtGui.QMainWindow):

```
def __init__(self):
    super(QtGui.QMainWindow, self).__init__()
    self.listItems = {}

    myQWidget = QtGui.QWidget()
    myBoxLayout = QtGui.QVBoxLayout()
    myQWidget.setLayout(myBoxLayout)
    self.setCentralWidget(myQWidget)

    self.listWidgetA = ThumbListWidget(self)
    for i in range(5):
        QtGui.QListWidgetItem('Item ' + str(i + 1), self.listWidgetA)

    for item in self.listWidgetA.iterAllItems():
        item.setFlags(item.flags() | Qt.ItemIsUserCheckable)
        item.setCheckState(Qt.Checked)

    myBoxLayout.addWidget(self.listWidgetA)
    self.listWidgetA.setAcceptDrops(False)
    self.listWidgetA.viewport().update()
```

if **name** == '**main**': app = QtGui.QApplication(sys.argv) dialog = Dialog() dialog.show() dialog.resize(400, 140) sys.exit(app.exec_())

但是还存在一个问题，希望能够在框选了多个item之后，通过单击任意一个item的checkbox，也能达到勾选/去选所有it

```python
    def mouseReleaseEvent(self, event):
        item = self.selectedCheckStateItem(event.pos())
        if item:
            selectedItems = self.selectedItems()
            new_state = Qt.Unchecked if item.checkState() == Qt.Checked else Qt.Checked
            self.setSelectedCheckStates(new_state, item)
            # QtGui.QApplication.processEvents()
            self.viewport().update()

        QtGui.QListWidget.mouseReleaseEvent(self, event)
        if item:
            for sel_item in selectedItems:
                sel_item.setSelected(True)

    def setSelectedCheckStates(self, state, click_item):
        for item in self.selectedItems():
            if item is not click_item:
                item.setCheckState(state)

    def selectedCheckStateItem(self, pos):
        item = self.itemAt(pos)
        if item:
            opt = QtGui.QStyleOptionButton()
            opt.rect = self.visualItemRect(item)
            rect = self.style().subElementRect(QtGui.QStyle.SE_ViewItemCheckIndicator, opt)
            if item in self.selectedItems() and rect.contains(pos):
                return item
        return None
```

其原理是当鼠标按键释放时，通过在 selectedCheckStateItem 中判断释放位置是否刚好在某个item的左侧的checkbox上，如果是，则返回此item，否则返回None。

如果确实鼠标按键在某个item左侧的checkbox上释放了，那就拿到他现在的勾选状态，然后相应的勾选/去选当前所有选中的items。

Note:

- 解决问题的关键点是

  ```
  opt = QtGui.QStyleOptionButton()
  opt.rect = self.visualItemRect(item)
  rect = self.style().subElementRect(QtGui.QStyle.SE_ViewItemCheckIndicator, opt)
  ```

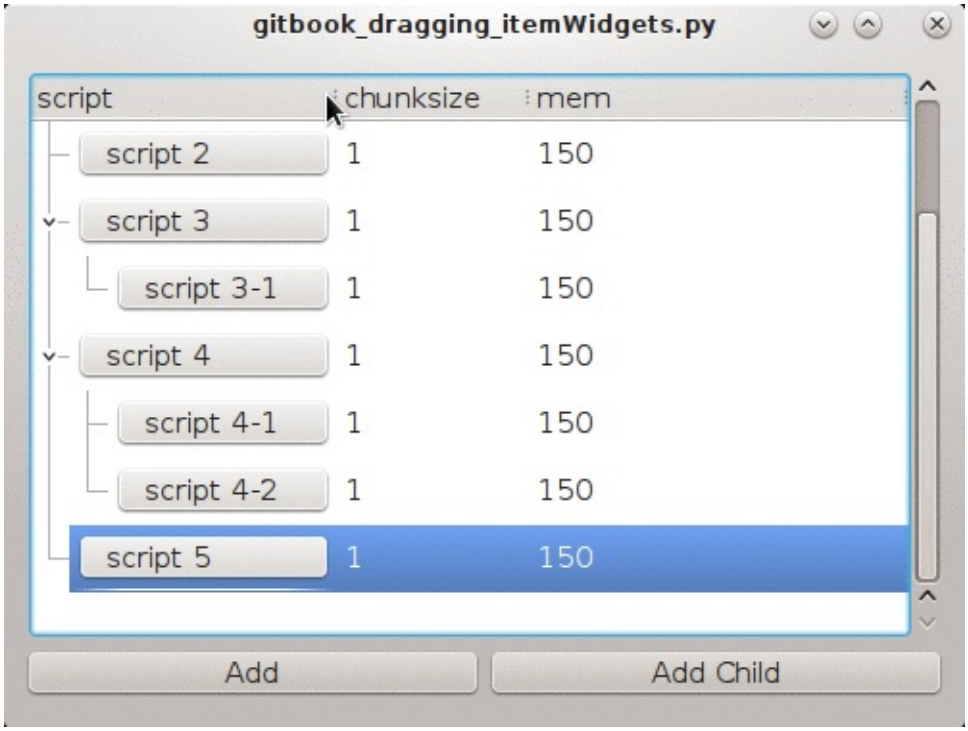
  此3行代码得到的是某个item的左侧的checkbox所占据的rect。

Tree

拖拽带itemWidget的treeWidgetItem

如果你在QTreeWidget的item上设置了itemWidget，你会发现拖放之后,itemWidget就消失了，这是“正常现象“，因为按照qt文档的描述，这个 `setItemWidget` 只能用来显示静态widet。

这里想了一个绕过的方法：给每个custom widget都写一个 `clone` method,使其 返回一个和自己当前状态完全一样的新的instance，然后在TreeWidget的dropEvent里调用这个 `clone` method，drop之前把"clone"出来的itemWidget存在list里，drop之后再使用刚才保存的itemWidget，对"clone"出来的itemWidget进行setItemWidget。



下面的代码包含了前一节的自定义drop indicator的效果

```
#!/usr/bin/env python2
import os
import sys
import re

from PyQt4 import QtGui, QtCore
from PyQt4.QtCore import Qt, QString


class MyWidget(QtGui.QDialog):

    def __init__(self, parent=None, val=None):
        super(MyWidget, self).__init__()
        self.layout = QtGui.QHBoxLayout(self)
        browseBtn = ElideButton(parent)
        browseBtn.setMinimumSize(QtCore.QSize(0, 25))
        browseBtn.setText(QString(val))
        browseBtn.setStyleSheet("text-align: left")
        self.layout.addWidget(browseBtn)
        self.browseBtn = browseBtn
        self.browseBtn.clicked.connect(self.browseCommandScript)
        self.browseBtn.setIconSize(QtCore.QSize(64, 64))

    def browseCommandScript(self):
```

```
        script = QtGui.QFileDialog.getOpenFileName(
            self, 'Select Script file', '/tmp/crap', "Executable Files (*)")
        if script:
            self._script = script
            old_text = str(self.browseBtn.text()).strip()
            old_text = re.search('^script [\d-]*', old_text).group()
            self.browseBtn.setText(('%s %s' % (old_text, script)))

    def clone(self):
        clone = MyWidget(val=str(self.browseBtn.text()))
        return clone


class ElideButton(QtGui.QPushButton):

    def __init__(self, parent=None):

        super(ElideButton, self).__init__(parent)
        font = self.font()
        font.setPointSize(10)
        self.setFont(font)

    def paintEvent(self, event):
        painter = QtGui.QStylePainter(self)

        metrics = QtGui.QFontMetrics(self.font())
        elided = metrics.elidedText(self.text(), Qt.ElideRight, self.width())

        option = QtGui.QStyleOptionButton()
        self.initStyleOption(option)
        option.text = ''
        painter.drawControl(QtGui.QStyle.CE_PushButton, option)
        painter.drawText(self.rect(), Qt.AlignLeft | Qt.AlignVCenter, elided)


class MyTreeView(QtGui.QTreeView):

    def __init__(self, parent=None):
        super(MyTreeView, self).__init__(parent)
        self.dropIndicatorRect = QtCore.QRect()

    def paintEvent(self, event):
        painter = QtGui.QPainter(self.viewport())
        self.drawTree(painter, event.region())
        # in original implementation, it calls an inline function paintDropIndicator here
        self.paintDropIndicator(painter)

    def paintDropIndicator(self, painter):

        if self.state() == QtGui.QAbstractItemView.DraggingState:
            opt = QtGui.QStyleOption()
            opt.init(self)
            opt.rect = self.dropIndicatorRect
            rect = opt.rect

            if rect.height() == 0:
                pen = QtGui.QPen(QtCore.Qt.black, 1, QtCore.Qt.DashLine)
                painter.setPen(pen)
                painter.drawLine(rect.topLeft(), rect.topRight())
            else:
                pen = QtGui.QPen(QtCore.Qt.black, 1, QtCore.Qt.DashLine)
                painter.setPen(pen)
                painter.drawRect(rect)
```

```
class MyTreeWidget(QtGui.QTreeWidget, MyTreeView):

    # def mouseMoveEvent(self, e):
    #     if self.state()==QtGui.QAbstractItemView.DraggingState:
    #         mimeData = self.model().mimeData(self.selectedIndexes())
    #         drag = QtGui.QDrag(self)
    #         drag.setMimeData(mimeData)
    #         drag.exec_(QtCore.Qt.MoveAction)

    def startDrag(self, supportedActions):
        listsQModelIndex = self.selectedIndexes()
        if listsQModelIndex:
            mimeData = QtCore.QMimeData()
            dataQMimeData = self.model().mimeData(listsQModelIndex)
            # if not dataQMimeData:
            #     return None
            dragQDrag = QtGui.QDrag(self)
            # dragQDrag.setPixmap(QtGui.QPixmap('test.jpg')) # <- For put your custom image here
            dragQDrag.setMimeData(dataQMimeData)
            defaultDropAction = QtCore.Qt.IgnoreAction
            if ((supportedActions & QtCore.Qt.CopyAction) and (self.dragDropMode() != QtGui.QAbstractItemV
                defaultDropAction = QtCore.Qt.CopyAction
            dragQDrag.exec_(supportedActions, defaultDropAction)

    def dragMoveEvent(self, event):
        pos = event.pos()
        item = self.itemAt(pos)

        if item:
            index = self.indexFromItem(item)

            rect = self.visualRect(index)
            rect_left = self.visualRect(index.sibling(index.row(), 0))
            rect_right = self.visualRect(index.sibling(index.row(), self.columnCount() - 1))
            self.dropIndicatorPosition = self.position(event.pos(), rect, index)

            if self.dropIndicatorPosition == self.AboveItem:
                self.dropIndicatorRect = QtCore.QRect(rect_left.left(), rect_left.top(), rect_right.right() - rect_lef
                event.accept()
            elif self.dropIndicatorPosition == self.BelowItem:
                self.dropIndicatorRect = QtCore.QRect(rect_left.left(), rect_left.bottom(), rect_right.right() - rec
                event.accept()

            elif self.dropIndicatorPosition == self.OnItem:
                self.dropIndicatorRect = QtCore.QRect(rect_left.left(), rect_left.top(), rect_right.right() - rect_lef
                event.accept()
            else:
                self.dropIndicatorRect = QtCore.QRect()

            self.model().setData(index, self.dropIndicatorPosition, Qt.UserRole)

            # self.setState(QtGui.QAbstractItemView.DraggingState)
        # This is necessary or else the previously drawn rect won't be erased
        self.viewport().update()

    def iterativeChildren(self, nodes):
        results = []
        while True:
            newNodes = []
            if not nodes:
                break
```

```
            for node in nodes:
                results.append(node)
                for i in range(node.childCount()):
                    print 'newNodes:', newNodes
                    newNodes += [node.child(i)]
            nodes = newNodes
        results = nodes + results
        return results

    def keyPressEvent(self, event):
        'delete currently selected item'
        QtGui.QTreeWidget.keyPressEvent(self, event)
        key = event.key()

        if self.currentItem():

            root = self.invisibleRootItem()
            parent = self.currentItem().parent() or root

            if key == Qt.Key_Delete:
                parent.removeChild(self.currentItem())

    def dropEvent(self, event):
        pos = event.pos()
        item = self.itemAt(pos)
        if item:
            index = self.indexFromItem(item)
            self.model().setData(index, 0, Qt.UserRole)

        if item is self.currentItem():
            QtGui.QTreeWidget.dropEvent(self, event)
            event.accept()
            return

        if event.source == self and event.dropAction() == Qt.MoveAction or self.dragDropMode() == QtGui.

            topIndex = QtCore.QModelIndex()
            col = -1
            row = -1

            l = [event, row, col, topIndex]

            if self.dropOn(l):

                event, row, col, topIndex = l

                idxs = self.selectedIndexes()
                indexes = []
                existing_rows = set()
                for i in idxs:
                    if i.row() not in existing_rows:
                        indexes.append(i)
                        existing_rows.add(i.row())

                if topIndex in indexes:
                    return

                # try storing the itemWidgets first
                # we should iterate through all child items,and store itemWidgets for them
                widgets = []

                dropRow = self.model().index(row, col, topIndex)
                taken = []
```

```
        indexes_reverse = indexes[:]
        indexes_reverse.reverse()
        # i = 0
        for index in indexes_reverse:
            parent = self.itemFromIndex(index)
            item_widget = self.itemWidget(parent, 0)

            print 'item_widget:', item_widget, item_widget.parent()

            # item_widget.setParent(self)
            print 'dragging item has child:', parent.childCount()

            # print 'before dragging, child 0 ',self.itemWidget( parent.child(0),0).browseBtn.text()

            # in case it has children , we get all of them
            all_child = []

            all_items = self.iterativeChildren([parent])

            print 'all items:', len(all_items), all_items

            # store cloned widgets in a list
            widgets = [self.itemWidget(i, 0).clone() for i in all_items]

            # widgets.append(item_widget.clone())

            if not parent or not parent.parent():
                # if not parent or not isinstance(parent.parent(),QtGui.QTreeWidgetItem):
                taken.append(self.takeTopLevelItem(index.row()))
            else:
                taken.append(parent.parent().takeChild(index.row()))

            # i += 1
            # break

        taken.reverse()

        print 'itemWidgets:', widgets

        for index in indexes:
            print 'inserting: topIndex:', topIndex.isValid(), row
            if row == -1:  # means index=root
                if topIndex.isValid():  # Returns the model index of the model's root item. The root item is
                    parent = self.itemFromIndex(topIndex)
                    parent.insertChild(parent.childCount(), taken[0])

                    # after insert the itemwidget is gone
                    # print 'after dragging, child 0 ',self.itemWidget( taken[0],0).browseBtn.text()

                    # self.setItemWidget(taken[0],0,QtGui.QLineEdit())
                    # self.setItemWidget(taken[0],0,new_widget)
                    print 'row==-1,if',  # self.itemWidget(taken[0],0),self.itemWidget(taken[0],0).parent()
                    # taken = taken[1:]

                else:
                    self.insertTopLevelItem(self.topLevelItemCount(), taken[0])
                    # taken = taken[1:]
                    print 'row==-1,else'
            else:
                r = dropRow.row() if dropRow.row() >= 0 else row
                if topIndex.isValid():
                    parent = self.itemFromIndex(topIndex)
```

```
                    parent.insertChild(min(r, parent.childCount()), taken[0])
                    # taken = taken[1:]
                    print 'row!=-1,if'
                else:
                    self.insertTopLevelItem(min(r, self.topLevelItemCount()), taken[0])
                    # taken = taken[1:]
                    print 'row!=-1,else'

            all_items = self.iterativeChildren([taken[0]])

            for i, w in zip(all_items, widgets):
                self.setItemWidget(i, 0, w)

            taken = taken[1:]
        event.accept()

    QtGui.QTreeWidget.dropEvent(self, event)
    self.expandAll()
    self.updateGeometry()

def position(self, pos, rect, index):
    r = QtGui.QAbstractItemView.OnViewport
    # margin*2 must be smaller than row height, or the drop onItem rect won't show
    margin = 10
    if pos.y() - rect.top() < margin:
        r = QtGui.QAbstractItemView.AboveItem
    elif rect.bottom() - pos.y() < margin:
        r = QtGui.QAbstractItemView.BelowItem
    # elif rect.contains(pos, True):
    elif pos.y() - rect.top() > margin and rect.bottom() - pos.y() > margin:
        r = QtGui.QAbstractItemView.OnItem

    return r

def dropOn(self, l):

    event, row, col, index = l

    root = self.rootIndex()

    if self.viewport().rect().contains(event.pos()):
        index = self.indexAt(event.pos())
        # if drop on nothing or drop out side of index zone
        print 'in drop on ', index, index.isValid(), self.visualRect(index).contains(event.pos())
        if not index.isValid() or not self.visualRect(index).contains(event.pos()):
            index = root

    if index != root:

        dropIndicatorPosition = self.position(event.pos(), self.visualRect(index), index)
        if self.dropIndicatorPosition == self.AboveItem:
            print 'dropon above'
            row = index.row()
            col = index.column()
            index = index.parent()

        elif self.dropIndicatorPosition == self.BelowItem:
            print 'dropon below'
            row = index.row() + 1
            col = index.column()
            index = index.parent()

        elif self.dropIndicatorPosition == self.OnItem:
```

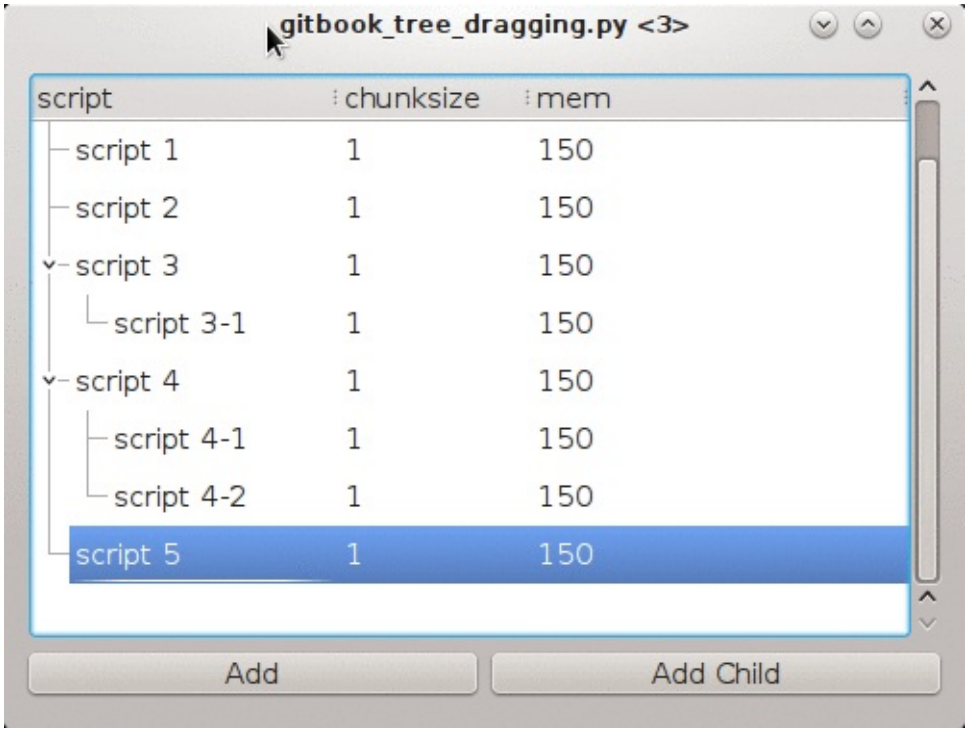
```
                print 'dropon onItem'
                pass
            elif self.dropIndicatorPosition == self.OnViewport:
                pass
            else:
                pass

        else:
            self.dropIndicatorPosition = self.OnViewport

        l[0], l[1], l[2], l[3] = event, row, col, index

        # if not self.droppingOnItself(event, index):
        return True


class TheUI(QtGui.QDialog):

    def __init__(self, args=None, parent=None):
        super(TheUI, self).__init__(parent)
        self.layout1 = QtGui.QVBoxLayout(self)
        treeWidget = MyTreeWidget()

        # treeWidget.setSelectionMode(QtGui.QAbstractItemView.ExtendedSelection)
        # treeWidget.setSelectionRectVisible(True)

        button1 = QtGui.QPushButton('Add')
        button2 = QtGui.QPushButton('Add Child')

        self.layout1.addWidget(treeWidget)

        self.layout2 = QtGui.QHBoxLayout()
        self.layout2.addWidget(button1)
        self.layout2.addWidget(button2)

        self.layout1.addLayout(self.layout2)

        treeWidget.setHeaderHidden(True)

        self.treeWidget = treeWidget
        self.button1 = button1
        self.button2 = button2
        self.button1.clicked.connect(lambda *x: self.addCmd())
        self.button2.clicked.connect(lambda *x: self.addChildCmd())

        HEADERS = ("script", "chunksize", "mem")
        self.treeWidget.setHeaderLabels(HEADERS)
        self.treeWidget.setColumnCount(len(HEADERS))

        self.treeWidget.setColumnWidth(0, 160)
        self.treeWidget.header().show()

        self.treeWidget.setDragDropMode(QtGui.QAbstractItemView.InternalMove)
        self.treeWidget.setStyleSheet('''
                                QTreeView {
                                    show-decoration-selected: 1;
                                }

                                QTreeView::item:hover {
                                    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #e7effd, stop
                                }

                                QTreeView::item:selected:active{
```

```
                                background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #6ea1f1, sto
                            }

                            QTreeView::item:selected:!active {
                                background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #6b9be8, sto
                            }
                            ''')

        self.resize(500, 350)
        for i in xrange(6):
            item = self.addCmd(i)
            if i in (3, 4):
                self.addChildCmd()
                if i == 4:
                    self.addCmd('%s-2' % i, parent=item)

        self.treeWidget.expandAll()
        self.setStyleSheet("QTreeWidget::item{ height: 30px;  }")

    def addChildCmd(self):
        parent = self.treeWidget.currentItem()
        self.addCmd(parent=parent)
        self.treeWidget.setCurrentItem(parent)

    def addCmd(self, i=None, parent=None):
        'add a level to tree widget'

        root = self.treeWidget.invisibleRootItem()
        if not parent:
            parent = root

        if i is None:
            if parent == root:
                i = self.treeWidget.topLevelItemCount()
            else:
                i = str(parent.text(0)).strip()[7:]
                i = '%s-%s' % (i, parent.childCount() + 1)

        # item = QtGui.QTreeWidgetItem(parent, ['script %s' % i, '1', '150'])

        script = '   script %s' % i
        item = QtGui.QTreeWidgetItem(parent, [script, '1', '150'])

        self.treeWidget.setItemWidget(item, 0, MyWidget(val=script))

        self.treeWidget.setCurrentItem(item)
        self.treeWidget.expandAll()
        return item

if __name__ == '__main__':
    app = QtGui.QApplication(sys.argv)
    gui = TheUI()
    gui.show()
    app.exec_()
```

自定义drop indicator

想实现如下的效果:

- 加粗的drop indicator
- 修改插入的“判定”灵敏度(这一点很重要，因为默认判定是2px，很难轻松的把拖拽的item“插入”两行之间)



其中

- `MyTreeView`里的`paintDropIndicator`用来自定义paint drop indicator
- `position` function用来修改默认的插入“判定”，原始默认值是2,显然`margin*2`必须小于行高，不然“恰好”放在item上的判定就没法发生了
- 在`dragMoveEvent`里通过position返回的“判定”，来决定表示放手位置的dropIndicatorRect的坐标
- `dropEvent`就是把c++版直接翻译了下，应该需要继续改进，很多地方不是python里的恰当写法

代码如下

```
#!/usr/bin/env python2

import os
import sys
import re

from PyQt4 import QtGui, QtCore
from PyQt4.QtCore import Qt, QString


class MyTreeView(QtGui.QTreeView):

    def __init__(self, parent=None):
        super(MyTreeView, self).__init__(parent)
        self.dropIndicatorRect = QtCore.QRect()

    def paintEvent(self, event):
```

```
        painter = QtGui.QPainter(self.viewport())
        self.drawTree(painter, event.region())
        # in original implementation, it calls an inline function paintDropIndicator here
        self.paintDropIndicator(painter)

    def paintDropIndicator(self, painter):

        if self.state() == QtGui.QAbstractItemView.DraggingState:
            opt = QtGui.QStyleOption()
            opt.init(self)
            opt.rect = self.dropIndicatorRect
            rect = opt.rect

            brush = QtGui.QBrush(QtGui.QColor(Qt.black))

            if rect.height() == 0:
                pen = QtGui.QPen(brush, 2, QtCore.Qt.SolidLine)
                painter.setPen(pen)
                painter.drawLine(rect.topLeft(), rect.topRight())
            else:
                pen = QtGui.QPen(brush, 2, QtCore.Qt.SolidLine)
                painter.setPen(pen)
                painter.drawRect(rect)


class MyTreeWidget(QtGui.QTreeWidget, MyTreeView):

    def startDrag(self, supportedActions):
        listsQModelIndex = self.selectedIndexes()
        if listsQModelIndex:
            mimeData = QtCore.QMimeData()
            dataQMimeData = self.model().mimeData(listsQModelIndex)
            dragQDrag = QtGui.QDrag(self)
            # dragQDrag.setPixmap(QtGui.QPixmap('test.jpg')) # <- For put your custom image here
            dragQDrag.setMimeData(dataQMimeData)
            defaultDropAction = QtCore.Qt.IgnoreAction
            if ((supportedActions & QtCore.Qt.CopyAction) and (self.dragDropMode() != QtGui.QAbstractItemV
                defaultDropAction = QtCore.Qt.CopyAction
            dragQDrag.exec_(supportedActions, defaultDropAction)

    def dragMoveEvent(self, event):
        pos = event.pos()
        item = self.itemAt(pos)

        if item:
            index = self.indexFromItem(item)  # this always get the default 0 column index

            rect = self.visualRect(index)
            rect_left = self.visualRect(index.sibling(index.row(), 0))
            rect_right = self.visualRect(index.sibling(index.row(), self.header().logicalIndex(self.columnCount(

            self.dropIndicatorPosition = self.position(event.pos(), rect, index)

            if self.dropIndicatorPosition == self.AboveItem:
                self.dropIndicatorRect = QtCore.QRect(rect_left.left(), rect_left.top(), rect_right.right() - rect_lef
                event.accept()
            elif self.dropIndicatorPosition == self.BelowItem:
                self.dropIndicatorRect = QtCore.QRect(rect_left.left(), rect_left.bottom(), rect_right.right() - rec
                event.accept()
            elif self.dropIndicatorPosition == self.OnItem:
                self.dropIndicatorRect = QtCore.QRect(rect_left.left(), rect_left.top(), rect_right.right() - rect_lef
                event.accept()
            else:
```

```
                self.dropIndicatorRect = QtCore.QRect()

            self.model().setData(index, self.dropIndicatorPosition, Qt.UserRole)

        # This is necessary or else the previously drawn rect won't be erased
        self.viewport().update()

    def dropEvent(self, event):
        pos = event.pos()
        item = self.itemAt(pos)

        if item is self.currentItem():
            QtGui.QTreeWidget.dropEvent(self, event)
            event.accept()
            return

        if item:
            index = self.indexFromItem(item)
            self.model().setData(index, 0, Qt.UserRole)

        if event.source == self and event.dropAction() == Qt.MoveAction or self.dragDropMode() == QtGui.

            topIndex = QtCore.QModelIndex()
            col = -1
            row = -1

            l = [event, row, col, topIndex]

            if self.dropOn(l):

                event, row, col, topIndex = l

                idxs = self.selectedIndexes()
                indexes = []
                existing_rows = set()
                for i in idxs:
                    if i.row() not in existing_rows:
                        indexes.append(i)
                        existing_rows.add(i.row())

                if topIndex in indexes:
                    return

                dropRow = self.model().index(row, col, topIndex)
                taken = []

                indexes_reverse = indexes[:]
                indexes_reverse.reverse()
                i = 0
                for index in indexes_reverse:
                    parent = self.itemFromIndex(index)
                    if not parent or not parent.parent():
                        # if not parent or not isinstance(parent.parent(),QtGui.QTreeWidgetItem):
                        taken.append(self.takeTopLevelItem(index.row()))
                    else:
                        taken.append(parent.parent().takeChild(index.row()))

                    i += 1
                    # break

                taken.reverse()

                for index in indexes:
```

```
                if row == -1:
                    if topIndex.isValid():
                        parent = self.itemFromIndex(topIndex)
                        parent.insertChild(parent.childCount(), taken[0])
                        taken = taken[1:]

                    else:
                        self.insertTopLevelItem(self.topLevelItemCount(), taken[0])
                        taken = taken[1:]
                else:
                    r = dropRow.row() if dropRow.row() >= 0 else row
                    if topIndex.isValid():
                        parent = self.itemFromIndex(topIndex)
                        parent.insertChild(min(r, parent.childCount()), taken[0])
                        taken = taken[1:]
                    else:
                        self.insertTopLevelItem(min(r, self.topLevelItemCount()), taken[0])
                        taken = taken[1:]

            event.accept()

        QtGui.QTreeWidget.dropEvent(self, event)
        self.expandAll()

    def position(self, pos, rect, index):
        r = QtGui.QAbstractItemView.OnViewport
        # margin*2 must be smaller than row height, or the drop onItem rect won't show
        margin = 10
        if pos.y() - rect.top() < margin:
            r = QtGui.QAbstractItemView.AboveItem
        elif rect.bottom() - pos.y() < margin:
            r = QtGui.QAbstractItemView.BelowItem

        # this rect is always the first column rect
        # elif rect.contains(pos, True):
        elif pos.y() - rect.top() > margin and rect.bottom() - pos.y() > margin:
            r = QtGui.QAbstractItemView.OnItem

        return r

    def dropOn(self, l):

        event, row, col, index = l

        root = self.rootIndex()

        if self.viewport().rect().contains(event.pos()):
            index = self.indexAt(event.pos())
            if not index.isValid() or not self.visualRect(index).contains(event.pos()):
                index = root

        if index != root:

            dropIndicatorPosition = self.position(event.pos(), self.visualRect(index), index)
            if self.dropIndicatorPosition == self.AboveItem:
                print 'dropon above'
                row = index.row()
                col = index.column()
                index = index.parent()

            elif self.dropIndicatorPosition == self.BelowItem:
                print 'dropon below'
                row = index.row() + 1
```

```
                col = index.column()
                index = index.parent()

            elif self.dropIndicatorPosition == self.OnItem:
                print 'dropon onItem'
                pass
            elif self.dropIndicatorPosition == self.OnViewport:
                pass
            else:
                pass

        else:
            self.dropIndicatorPosition = self.OnViewport

        l[0], l[1], l[2], l[3] = event, row, col, index

        # if not self.droppingOnItself(event, index):
        return True


class TheUI(QtGui.QDialog):

    def __init__(self, args=None, parent=None):
        super(TheUI, self).__init__(parent)
        self.layout1 = QtGui.QVBoxLayout(self)
        treeWidget = MyTreeWidget()

        treeWidget.setSelectionMode(QtGui.QAbstractItemView.ExtendedSelection)

        button1 = QtGui.QPushButton('Add')
        button2 = QtGui.QPushButton('Add Child')

        self.layout1.addWidget(treeWidget)

        self.layout2 = QtGui.QHBoxLayout()
        self.layout2.addWidget(button1)
        self.layout2.addWidget(button2)

        self.layout1.addLayout(self.layout2)

        treeWidget.setHeaderHidden(True)

        self.treeWidget = treeWidget
        self.button1 = button1
        self.button2 = button2
        self.button1.clicked.connect(lambda *x: self.addCmd())
        self.button2.clicked.connect(lambda *x: self.addChildCmd())

        HEADERS = ("script", "chunksize", "mem")
        self.treeWidget.setHeaderLabels(HEADERS)
        self.treeWidget.setColumnCount(len(HEADERS))

        self.treeWidget.setColumnWidth(0, 160)
        self.treeWidget.header().show()

        self.treeWidget.setDragDropMode(QtGui.QAbstractItemView.InternalMove)
        self.treeWidget.setStyleSheet('''
                                    QTreeView {
                                        show-decoration-selected: 1;
                                    }

                                    QTreeView::item:hover {
                                        background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #e7effd, stop
```

```
                                }

                                QTreeView::item:selected:active{
                                    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #6ea1f1, sto
                                }

                                QTreeView::item:selected:!active {
                                    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #6b9be8, sto
                                }
                                ''')

        self.resize(500, 350)
        for i in xrange(6):
            item = self.addCmd(i)
            if i in (3, 4):
                self.addChildCmd()
                if i == 4:
                    self.addCmd('%s-2' % i, parent=item)

        self.treeWidget.expandAll()
        self.setStyleSheet("QTreeWidget::item{ height: 30px;  }")

    def addChildCmd(self):
        parent = self.treeWidget.currentItem()
        self.addCmd(parent=parent)
        self.treeWidget.setCurrentItem(parent)

    def addCmd(self, i=None, parent=None):
        'add a level to tree widget'

        root = self.treeWidget.invisibleRootItem()
        if not parent:
            parent = root

        if i is None:
            if parent == root:
                i = self.treeWidget.topLevelItemCount()
            else:
                i = str(parent.text(0))[7:]
                i = '%s-%s' % (i, parent.childCount() + 1)

        item = QtGui.QTreeWidgetItem(parent, ['script %s' % i, '1', '150'])

        self.treeWidget.setCurrentItem(item)
        self.treeWidget.expandAll()
        return item

if __name__ == '__main__':
    app = QtGui.QApplication(sys.argv)
    gui = TheUI()
    gui.show()
    app.exec_()
```

Table

StyleSheet

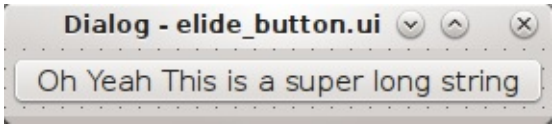
Designer

Promoted Widgets

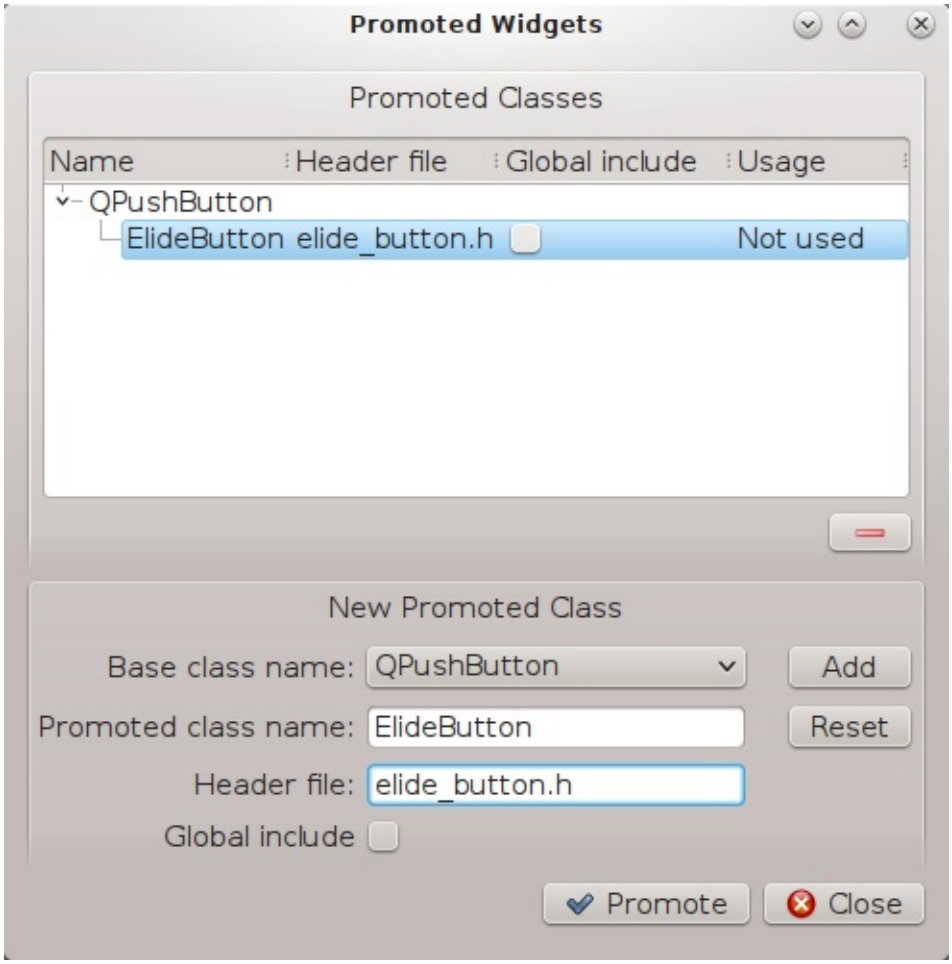
如果你有一些自定义的widgets,但是又希望用designer来画ui，这时有两种做法：

- 把这些widgets做成插件，让他们出现在designer左侧的widget列表里。但这样难度比较高，如果不是需要经常反复使用的widget，没必要这么做
- 使用promoted widgets，操作相对简单

以3.1带overflow效果的按钮为例：



如果你定义了ElideButton这个Subclass，你可以把他保存成单独一个文件elide_button.py，然后在designer里依然用QPushButton，但是创建之后在上面右击，选择Promote to,然后照下图填入



其中ElideButton必须和你的subclass的名字一致，elide_button.h必须和你的.py文件名一致(请无视.h)

这样你就可以达到，虽然designer里画的是QPushButton,但是当运行的时候，他其实使用的是你的ElideButton。

代码如下：

```
#!/usr/bin/env python2
import os
import sys
from PyQt4 import QtGui, QtCore
from PyQt4.QtCore import Qt, QString
from PyQt4 import uic


class TheUI(QtGui.QDialog):

    def __init__(self, args=None, parent=None):
        super(TheUI, self).__init__(parent)
        self.ui = uic.loadUi('elide_button.ui', self)
        self.pushButton.setText('Oh Yeah This is a super long string')
        self.ui.show()
        self.setMinimumWidth(20)

if __name__ == '__main__':
    app = QtGui.QApplication(sys.argv)
    gui = TheUI()
    gui.show()
    app.exec_()
```

综上，你可以随意使用自己定义的widget,与此同时还可以使用designer来画ui，只要你在designer里把相应的widget promote成你自定义的widget即可。

Graphics View