PyQt4 编程简介

作者:

柴树杉[翻译] (chaishushan@gmail.com)

日期:

2007-12-22 于武汉

注解:

该文档根据"Introduction to PyQt4"翻译，依照创作公用约定发布。

该文档的doxygen源文件可以从pyqt-doc-cn下载。

开始

创建一个 PyQt4 一般可以通过很少的步骤完成。通常的方法是用 Qt 提供的QtDesigner工具创建界面。使用QtDesigner，可以方便地创建复杂的GUI界面。然后，可以在窗口上创建部件， 添加名字等。创建一个PyQt4 一般需要：

1. 使用 QtDesigner 创建 GUI 界面
2. 在属性编辑器中修改部件的名字
3. 使用 pyuic4 工具生成一个 python 类
4. 通过 GUI 对应类来运行程序
5. 通过设置自己的 slots 来扩展功能
6. 当使用窗口部件的时候，可以从 "PyQt's Classes"查询。Qt采用易于理解的方式来命名函数，例如："setText"。

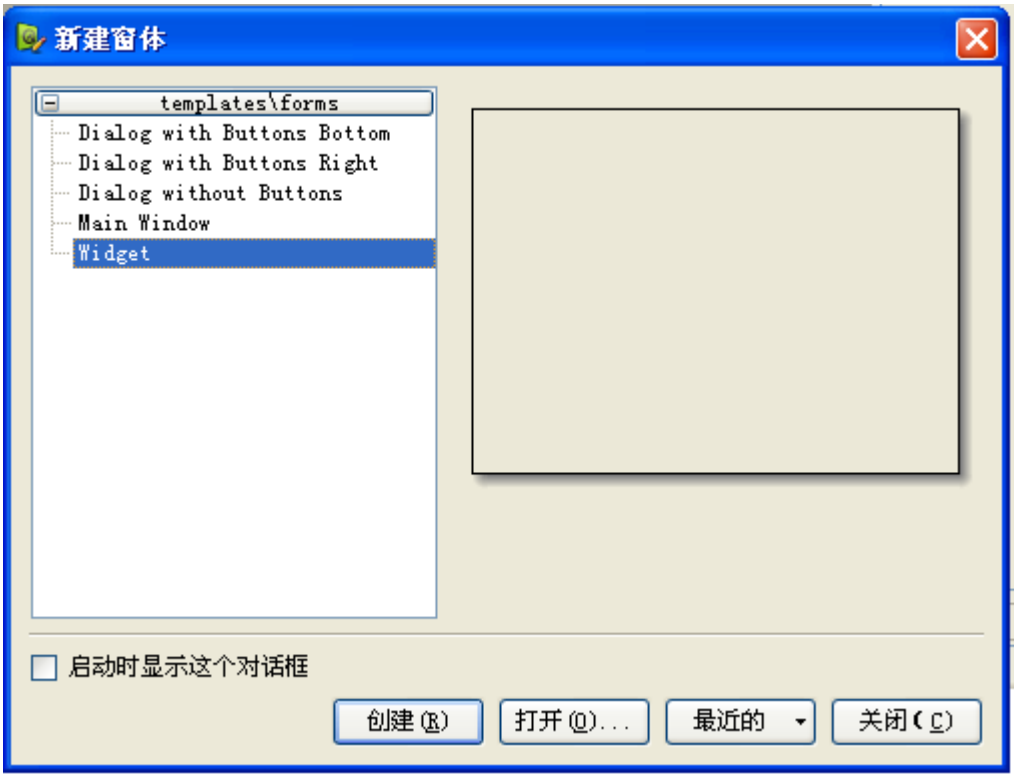
教程列表

1. **简易的文本编辑器** - PyQt4 第一个程序
2. **增加文本编辑器的功能** - 增加更多的功能
3. **QYolk I - PyQt4 中的列表部件** - 怎么使用PyQt4 中的列表部件
4. **QYolk II - 容器部件** - 怎么使用Tab Widget
5. **PyQt4 文本编辑器 - 最终版** - PyQt4 的一些高级特性
6. **QYolk III - 升级包列表** - 新的特性

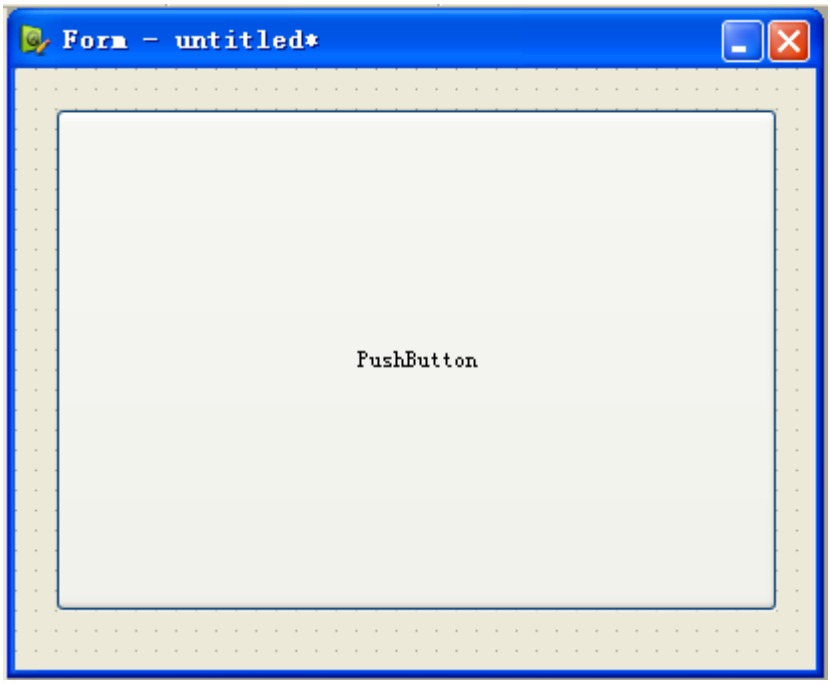
- 下载教程代码 (缺少的部分请从原网站下载)

简介

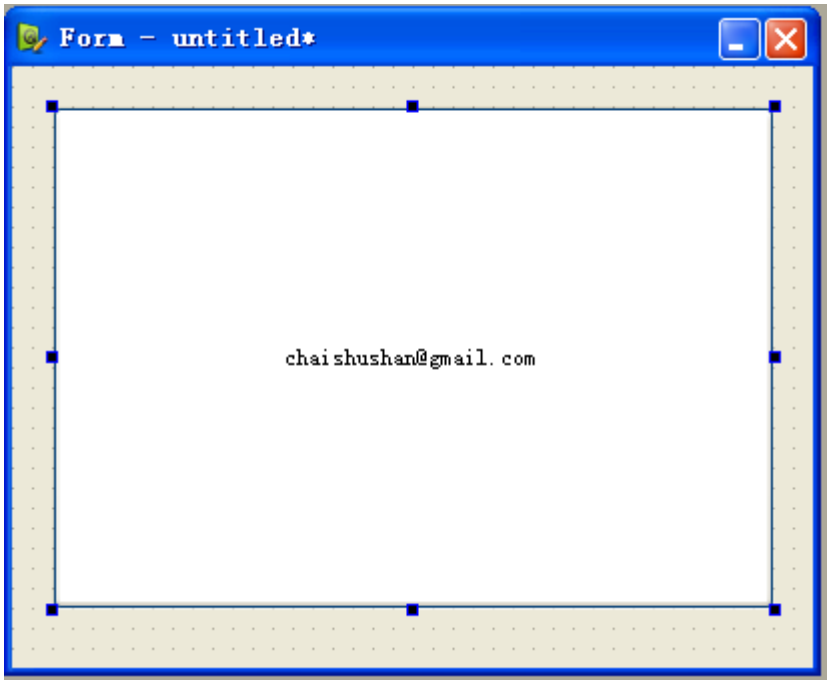
打开 QtDesigner，会出现"Hello... Close Button"对话框，让我们选择类型类型：



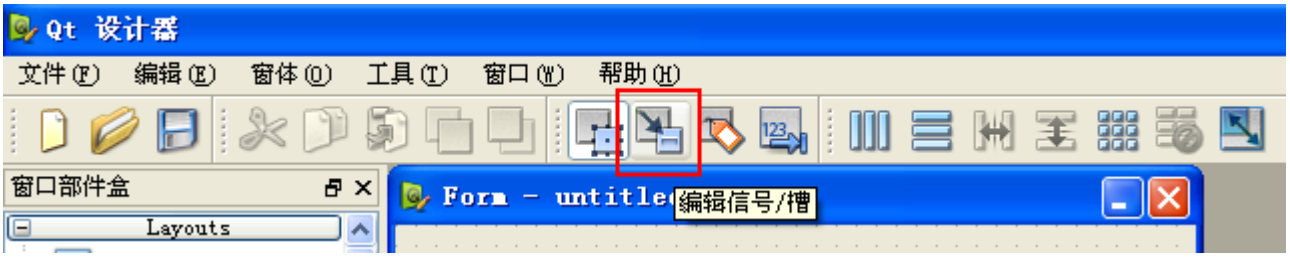
我们选择 widget 类型，然后在窗口中添加一个 PushButton 按钮：



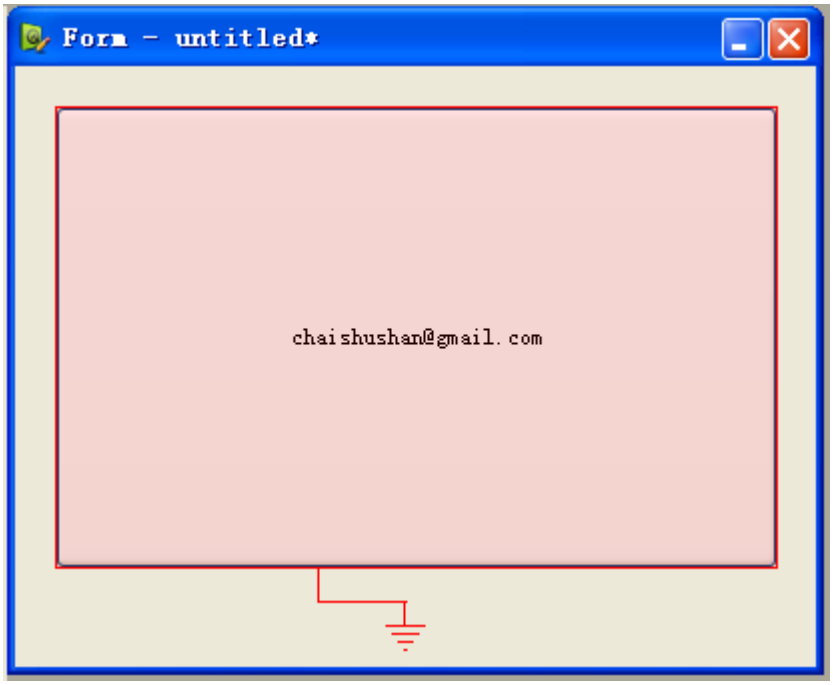
通过鼠标右键来修改 pushButton 显示的内容：



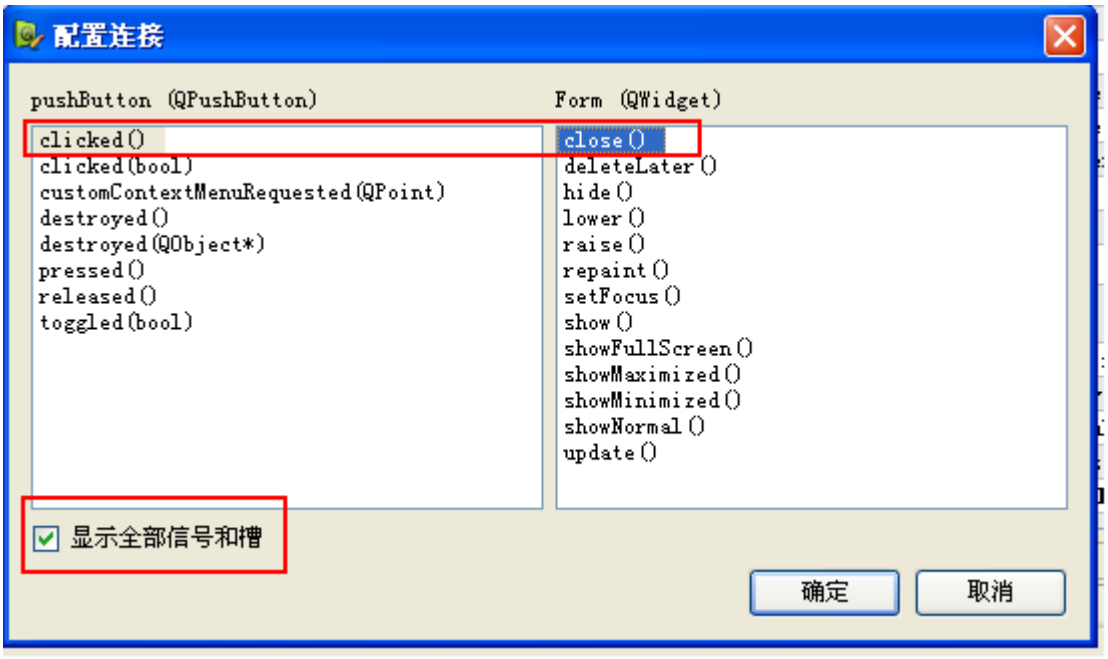
当窗口建好之后，我们可以 QtDesigner 来编辑一些 Qt 预定义的信号/槽。这里我们使用的是 "close()"槽函数 来关闭程序。首先切换到信号/槽边界模式：



用鼠标移到 pushButton 区域，然后拖动：



弹出一个信号/槽选择框：



信号选择 clicked()，槽选择 close()。将窗口保存为 test.ui 文件。切换到 test.ui 所在的目录，然后输入以下命令：

```
pyuic4 test.ui > test_ui.py
```

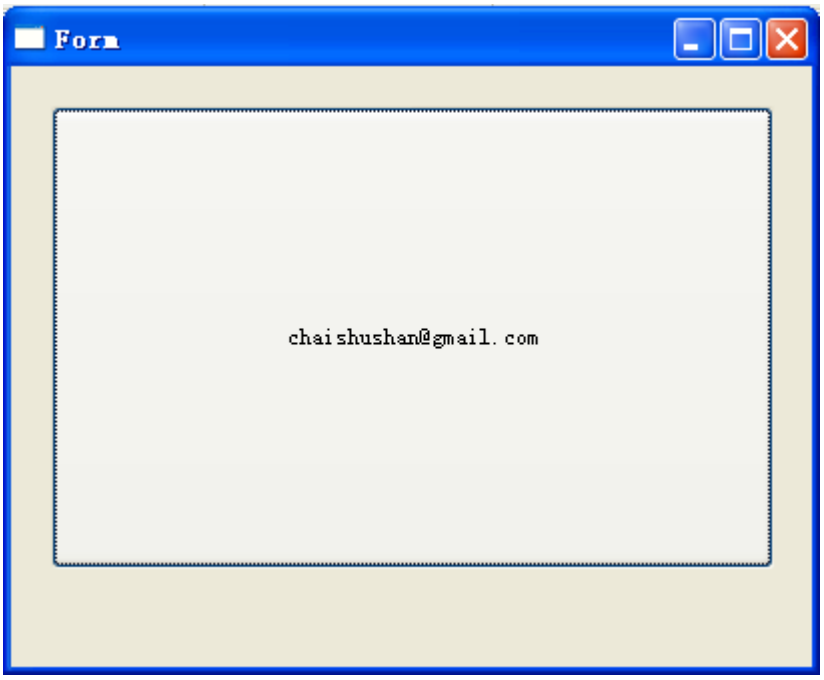
下一步是创建一个 test.py 文件：

```
import sys
from PyQt4 import QtCore, QtGui

from test_ui import Ui_Form

class MyForm(QtGui.QMainWindow):
    def __init__(self, parent=None):
        QtGui.QWidget.__init__(self, parent)
        self.ui = Ui_Form()
        self.ui.setupUi(self)

if __name__ == "__main__":
    app = QtGui.QApplication(sys.argv)
    myapp = MyForm()
    myapp.show()
    sys.exit(app.exec_())
```

运行 test.py：

```
python test.py
```

现在应该出现响应的窗口，当你点击按钮的时候退出程序。

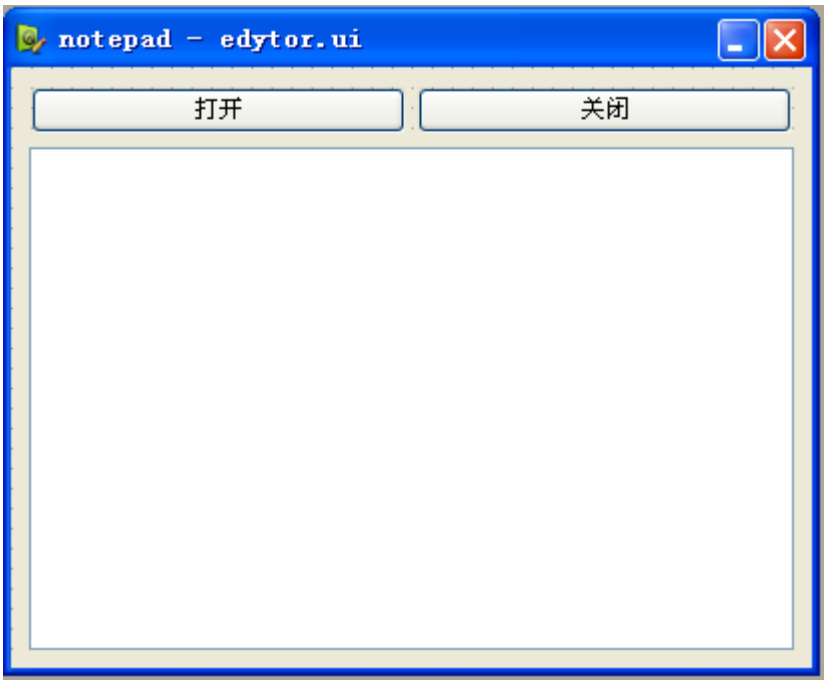


提示

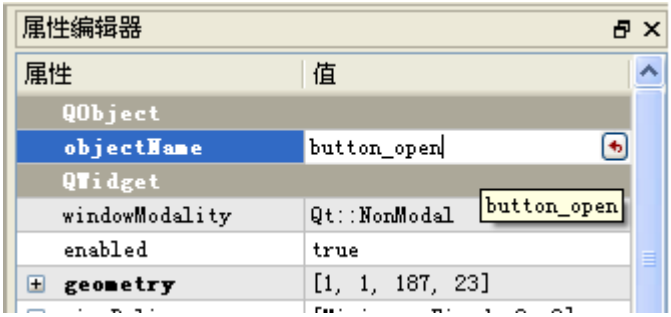
Ui_Form 是用 pyuic4 工具从"Form"窗口生成的对应 python 类的名字。你可以在 QtDesigner 自己喜欢的名字 一个类的名字（下一节我们会讲到）。

简易的文本编辑器

我们将要实现一个简单的文本编辑器，如图。用 QtDesigner 创建一个"Widget"类型的窗口。 我们使用两个 PushButton 和一个 TextEdit：



"关闭"按钮被连接到窗口的"close()"槽函数，可以被用来关闭窗口。修改"打开"按钮的对象名字为"button_open"；修改 TextEdit 部件的对象名字为"editor_window"；修改窗口的名字为"notepad"（开始为"MainWindow"）。选择要该名字的对象，然后出现的属性编辑器中可以修改名字。



保存窗口，并生成对应的类：

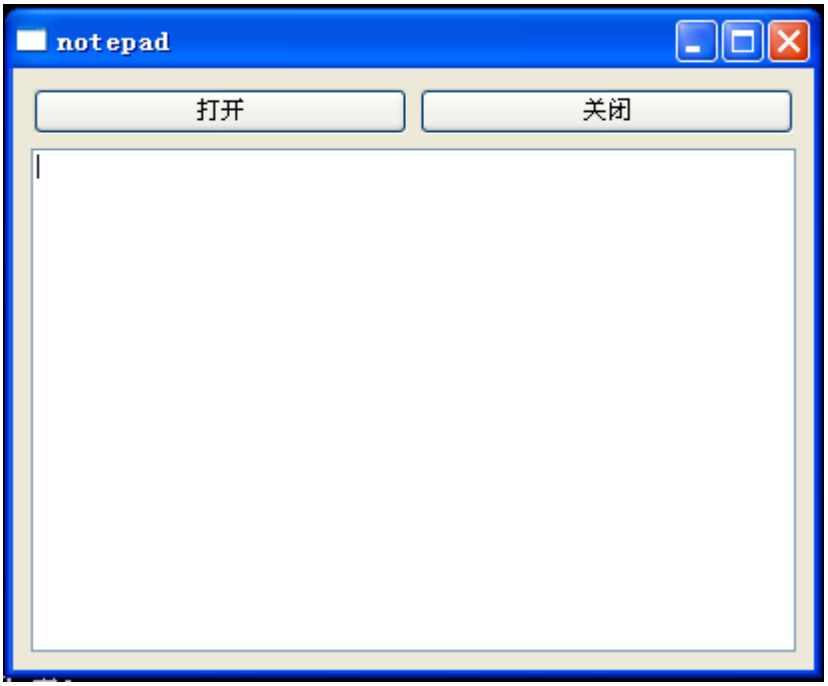
```
pyuic4 edytor.ui > edytor.py
```

得到一个"Ui_notepad"类。我们还需要自己添加一些代码，创建 start.py：

```
import sys
```

```
from PyQt4 import QtCore, QtGui
from edytor import Ui_notepad

class StartQt4(QtGui.QMainWindow):
    def __init__(self, parent=None):
        QtGui.QWidget.__init__(self, parent)
        self.ui = Ui_notepad()
        self.ui.setupUi(self)

if __name__ == "__main__":
    app = QtGui.QApplication(sys.argv)
    myapp = StartQt4()
    myapp.show()
    sys.exit(app.exec_())
```

运行 start.py 启动程序，点击“关闭”关闭程序。



下面我们编辑自己的 slot 函数：

```
import sys
from PyQt4 import QtCore, QtGui
from edytor import Ui_notepad

class StartQt4(QtGui.QMainWindow):
```

```
	def __init__(self, parent=None):
		QtGui.QWidget.__init__(self, parent)
		self.ui = Ui_notepad()
		self.ui.setupUi(self)
		# here we connect signals with our slots

QtCore.QObject.connect(self.ui.button_open,QtCore.SIGNAL("clicked()"),
self.file_dialog)
	def file_dialog(self):
		self.ui.editor_window.setText('aaaaaaaaaa')

if __name__ == "__main__":
	app = QtGui.QApplication(sys.argv)
	myapp = StartQt4()
	myapp.show()
	sys.exit(app.exec_())
```

当你点击“打开”的时候，在编辑框中将出现"aaaaaaaaaa"内容。那是因为我们把“打开”信号 连接到了我们自己实现的 slot 函数：

```
QtCore.QObject.connect(self.ui.button_open,QtCore.SIGNAL("clicked()"),
self.file_dialog)
```

self.ui 对应窗口，通过它我们可以访问窗口中的部件。因此，self.ui.button_open 对应“打开”按钮。self.file_dialog 是信号对应的函数，它是比较重要的部分，例如：

```
def file_dialog(self):
	self.ui.editor_window.setText('aaaaaaaaaa')
```

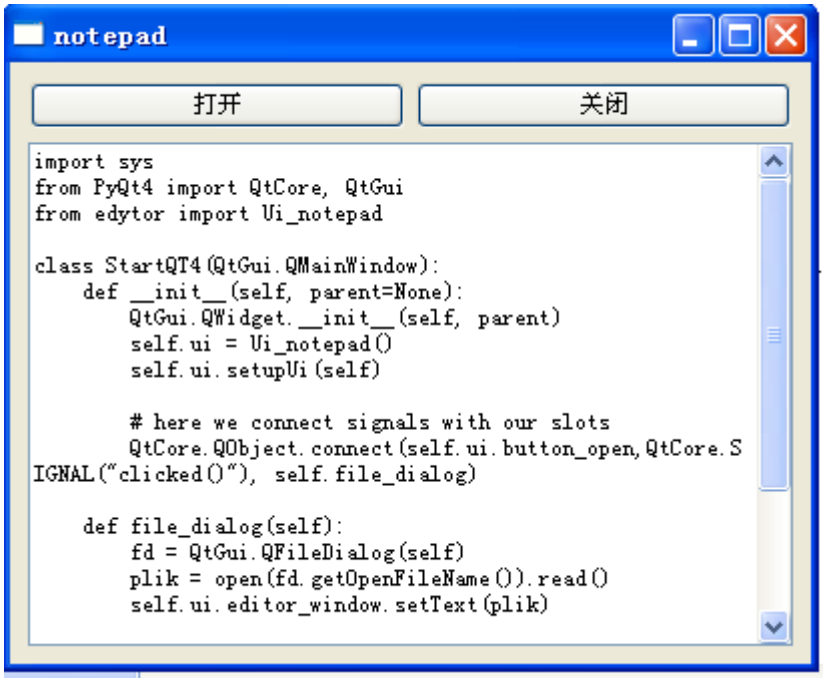
self.ui.editor_window 对应 TextEdit，setText 方法用来设置文本的内容。下面我们用 QFileDialog 来选择文件，代码如下：

```
import sys
from PyQt4 import QtCore, QtGui
from edytor import Ui_notepad

class StartQt4(QtGui.QMainWindow):
	def __init__(self, parent=None):
		QtGui.QWidget.__init__(self, parent)
		self.ui = Ui_notepad()
		self.ui.setupUi(self)
```

```
QtCore.QObject.connect(self.ui.button_open,QtCore.SIGNAL("clicked()"),
self.file_dialog)
       def file_dialog(self):
              fd = QtGui.QFileDialog(self)
              plik = open(fd.getOpenFileName()).read()
              self.ui.editor_window.setText(plik)

if __name__ == "__main__":
       app = QtGui.QApplication(sys.argv)
       myapp = StartQt4()
       myapp.show()
       sys.exit(app.exec_())
```

fd.getOpenFileName()弹出一个文件选择框。fd.getOpenFileName()用于返回我们选择文件的名字。 但是，如果我们没有选择文件的话，将得到一个空的文件名，程序出现以下错误：

```
IOError: [Errno 2] Nie ma takiego pliku ani katalogu: < PyQt4.QtCore.QString object
at 0x2b6465569738 >
```



继续完善代码：

So we have to improve our code:

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

```
from edytor import Ui_notepad

class StartQt4(QtGui.QMainWindow):
	def __init__(self, parent=None):
		QtGui.QWidget.__init__(self, parent)
		self.ui = Ui_notepad()
		self.ui.setupUi(self)
		# tutaj dajemy wlasne polaczenia slotow

QtCore.QObject.connect(self.ui.button_open,QtCore.SIGNAL("clicked()"),
self.file_dialog)
	def file_dialog(self):
		fd = QtGui.QFileDialog(self)
		self.filename = fd.getOpenFileName()
		from os.path import isfile
		if isfile(self.filename):
			text = open(self.filename).read()
			self.ui.editor_window.setText(text)

if __name__ == "__main__":
	app = QtGui.QApplication(sys.argv)
	myapp = StartQt4()
	myapp.show()
	sys.exit(app.exec_())
```

目前我们可以浏览文件了，但是并不能保存文件的任何修改。还需要增加一个“保存”按钮用来保存文件。 在 QtDesigner 中添加一个 pushButton，名字改为"button_save"，保存*.ui 文件。然后重新生成对应的类：

```
pyuic4 edytor.ui > edytor.py
```

现在程序外观如图：

我们给"Save"按钮的 click 信号连接对应的槽函数：

```
import sys
from PyQt4 import QtCore, QtGui
from edytor import Ui_notepad

class StartQt4(QtGui.QMainWindow):
	def __init__(self, parent=None):
		QtGui.QWidget.__init__(self, parent)
		self.ui = Ui_notepad()
		self.ui.setupUi(self)

QtCore.QObject.connect(self.ui.button_open,QtCore.SIGNAL("clicked()"),
self.file_dialog)

QtCore.QObject.connect(self.ui.button_save,QtCore.SIGNAL("clicked()"),
self.file_save)
	def file_dialog(self):
		fd = QtGui.QFileDialog(self)
		self.filename = fd.getOpenFileName()
		from os.path import isfile
		if isfile(self.filename):
			text = open(self.filename).read()
			self.ui.editor_window.setText(text)
	def file_save(self):
```

```
            from os.path import isfile
            if isfile(self.filename):
                  file = open(self.filename, 'w')
                  file.write(self.ui.editor_window.toPlainText())
                  file.close()

if __name__ == "__main__":
      app = QtGui.QApplication(sys.argv)
      myapp = StartQt4()
      myapp.show()
      sys.exit(app.exec_())
```

现在程序虽然可以基本工作，但是并不完善！它只能处理 ASCII 格式的文件。因此，我们最好给它增加 UTF-8 格式的支持：

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

class StartQt4(QtGui.QMainWindow):
      def __init__(self, parent=None):
            QtGui.QWidget.__init__(self, parent)
            self.ui = Ui_notepad()
            self.ui.setupUi(self)

QtCore.QObject.connect(self.ui.button_open,QtCore.SIGNAL("clicked()"),
self.file_dialog)

QtCore.QObject.connect(self.ui.button_save,QtCore.SIGNAL("clicked()"),
self.file_save)
      def file_dialog(self):
            fd = QtGui.QFileDialog(self)
            self.filename = fd.getOpenFileName()
            from os.path import isfile
            if isfile(self.filename):
                  import codecs
                  s = codecs.open(self.filename,'r','utf-8').read()
                  self.ui.editor_window.setPlainText(s)
      def file_save(self):
```

```
            from os.path import isfile
            if isfile(self.filename):
                  import codecs
                  s = codecs.open(self.filename,'w','utf-8')
                  s.write(unicode(self.ui.editor_window.toPlainText()))
                  s.close()

if __name__ == "__main__":
      app = QtGui.QApplication(sys.argv)
      myapp = StartQt4()
      myapp.show()
      sys.exit(app.exec_())
```

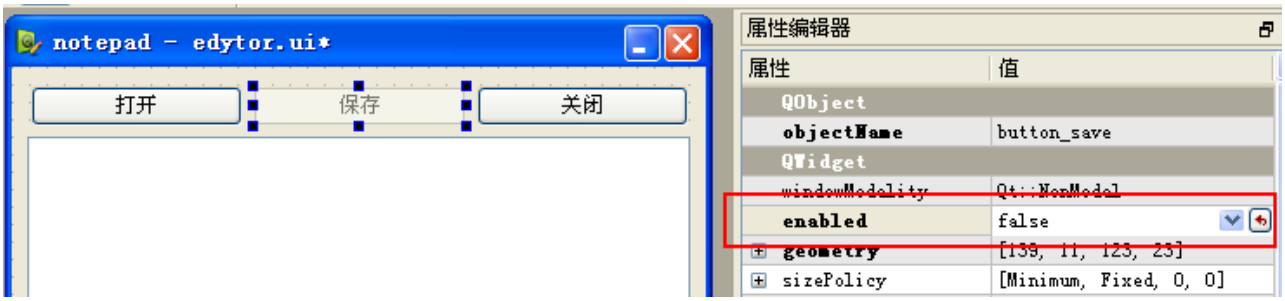
现在可以完美地处理 UTF-8 格式的文件了。下一节我们将演示更多的特性。

增加文本编辑器的功能

现在我们给编辑器增加两个新的功能。同时也可以练习我们查阅文档的技巧。

禁用"Save"按钮

当没有打开任何文件，或者是文件没有改动的时候，禁用"Save"按钮。在 QtDesigner 工具的属性编辑器中， 我们将"Save"按钮设置"enabled"属性为"False"。



textEdit 部件含有"textChanged()"信号，因此检测文本是否改动比较简单。但是，pushButton 并没有类似的 "enabled"操作。查阅文档可以发现：pushButton 从 QAbstractButton 继承，QAbstractButton 从 QWidget 继承， 而 QWidget 刚好有 setEnabled()函数。因此可以修改代码：

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

class StartQt4(QtGui.QMainWindow):
       def __init__(self, parent=None):
              QtGui.QWidget.__init__(self, parent)
              self.ui = Ui_notepad()
              self.ui.setupUi(self)
              self.filename = None

QtCore.QObject.connect(self.ui.button_open,QtCore.SIGNAL("clicked()"),
self.file_dialog)

QtCore.QObject.connect(self.ui.button_save,QtCore.SIGNAL("clicked()"),
self.file_save)

QtCore.QObject.connect(self.ui.editor_window,QtCore.SIGNAL("textChanged()"),
self.enable_save)
```

```
	def file_dialog(self):
		fd = QtGui.QFileDialog(self)
		self.filename = fd.getOpenFileName()
		from os.path import isfile
		if isfile(self.filename):
			import codecs
			s = codecs.open(self.filename,'r','utf-8').read()
			self.ui.editor_window.setPlainText(s)
			# inserting text emits textChanged() so we disable the
button :)
			self.ui.button_save.setEnabled(False)
	def enable_save(self):
		self.ui.button_save.setEnabled(True)
	def file_save(self):
		from os.path import isfile
		if isfile(self.filename):
			import codecs
			s = codecs.open(self.filename,'w','utf-8')
			s.write(unicode(self.ui.editor_window.toPlainText()))
			s.close()
			self.ui.button_save.setEnabled(False)

if __name__ == "__main__":
	app = QtGui.QApplication(sys.argv)
	myapp = StartQt4()
	myapp.show()
	sys.exit(app.exec_())
```

增加以下连接：

```
QtCore.QObject.connect(self.ui.editor_window,QtCore.SIGNAL("textChanged()"),
self.enable_save)
def enable_save(self):
	self.ui.button_save.setEnabled(True)
```

当我们修改文本的时候，file_dialog 函数将由"textChanged()"触发，因此我们在函数末尾应该禁止"Save"按钮：

```
self.ui.editor_window.setPlainText(s)
# inserting text emits textChanged() so we disable the button :)
```

```
self.ui.button_save.setEnabled(False)
```

保存修改

如果修改了文件没有保存的时候，又尝试打开新的文件，我们应该给出相关的提示信息。 可以使用 QMessageBox 提供的功能：

```
message = QtGui.QMessageBox(self)
message.exec_()
```

提示窗口还需要进一步改进，增加一些按钮和信息。首先修改 file_dialog 函数，如果文本没有被保存的时候 显示提示消息。但是怎么才能知道文本没有被保存呢？答案是"Save"没有被禁用的时候（self.ui.button_save.isEnabled()）。修改 start.py：

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

class StartQt4(QtGui.QMainWindow):
	def __init__(self, parent=None):
		QtGui.QWidget.__init__(self, parent)
		self.ui = Ui_notepad()
		self.ui.setupUi(self)
		self.filename = None

QtCore.QObject.connect(self.ui.button_open,QtCore.SIGNAL("clicked()"),
self.file_dialog)

QtCore.QObject.connect(self.ui.button_save,QtCore.SIGNAL("clicked()"),
self.file_save)

QtCore.QObject.connect(self.ui.editor_window,QtCore.SIGNAL("textChanged()"),
self.enable_save)
	def file_dialog(self):
		response = False
```

```
            # buttons texts
            SAVE = 'Save'
            DISCARD = 'Discard'
            CANCEL = 'Cancel'
            # if we have changes then ask about them
            if self.ui.button_save.isEnabled() and self.filename:
                  message = QtGui.QMessageBox(self)
                  message.setText('What to do about unsaved changes ?')
                  message.setWindowTitle('Notepad')
                  message.setIcon(QtGui.QMessageBox.Question)
                  message.addButton(SAVE, QtGui.QMessageBox.AcceptRole)
                  message.addButton(DISCARD,
QtGui.QMessageBox.DestructiveRole)
                  message.addButton(CANCEL, QtGui.QMessageBox.RejectRole)
                  message.setDetailedText('Unsaved changes in file: ' +
str(self.filename))
                  message.exec_()
                  response = message.clickedButton().text()
                  # save  file
                  if response == SAVE:
                        self.file_save()
                        self.ui.button_save.setEnabled(False)
                  # discard changes
                  elif response == DISCARD:
                        self.ui.button_save.setEnabled(False)
            # if we didn't cancelled show the file dialogue
            if response != CANCEL:
                  fd = QtGui.QFileDialog(self)
                  self.filename = fd.getOpenFileName()
                  from os.path import isfile
                  if isfile(self.filename):
                        import codecs
                        s = codecs.open(self.filename,'r','utf-8').read()
                        self.ui.editor_window.setPlainText(s)
                        self.ui.button_save.setEnabled(False)
      def enable_save(self):
            self.ui.button_save.setEnabled(True)
      def file_save(self):
            from os.path import isfile
```

```
            if isfile(self.filename):
                  import codecs
                  s = codecs.open(self.filename,'w','utf-8')
                  s.write(unicode(self.ui.editor_window.toPlainText()))
                  s.close()
                  self.ui.button_save.setEnabled(False)

if __name__ == "__main__":
      app = QtGui.QApplication(sys.argv)
      myapp = StartQt4()
      myapp.show()
      sys.exit(app.exec_())
```

新增加的代码如下：

```
response = False
# buttons texts
SAVE = 'Save'
DISCARD = 'Discard'
CANCEL = 'Cancel'
# if we have changes then ask about them
if self.ui.button_save.isEnabled() and self.filename:
      message = QtGui.QMessageBox(self)
      message.setText('What to do about unsaved changes ?')
      message.setWindowTitle('Notepad')
      message.setIcon(QtGui.QMessageBox.Question)
      message.addButton(SAVE, QtGui.QMessageBox.AcceptRole)
      message.addButton(DISCARD, QtGui.QMessageBox.DestructiveRole)
      message.addButton(CANCEL, QtGui.QMessageBox.RejectRole)
      message.setDetailedText('Unsaved changes in file: ' + str(self.filename))
      message.exec_()
      response = message.clickedButton().text()
      # save  file
      if response == SAVE:
            self.file_save()
            self.ui.button_save.setEnabled(False)
      # discard changes
      elif response == DISCARD:
            self.ui.button_save.setEnabled(False)
```

```
# if we didn't cancelled show the file dialogue
if response != CANCEL:
```

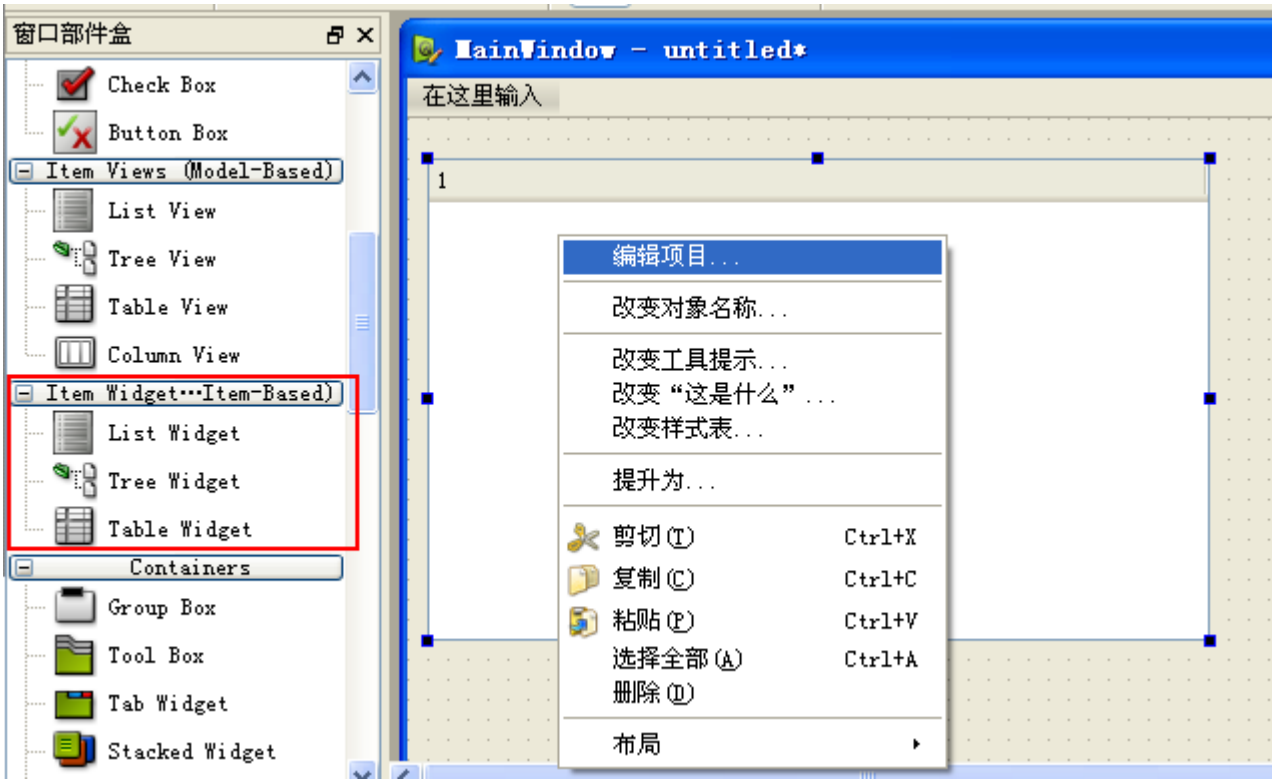
我们使用 QtGui.QMessageBox 生成信息提示框，然后设置文本信息、标题、图标、并且增件了三个按钮。 第二个参数设置每个按纽对应的 ButtonRole（具体细节可以参考文档）。setDetailedText 设置详细的 提示信息。然后通过 exec_()来运行消息提示框。消息框返回被按下的按纽值，因此我们可以根据返回 值来选择下一步要进行的操作。消息框外观如下：

QYolk I - PyQt4 中的列表部件

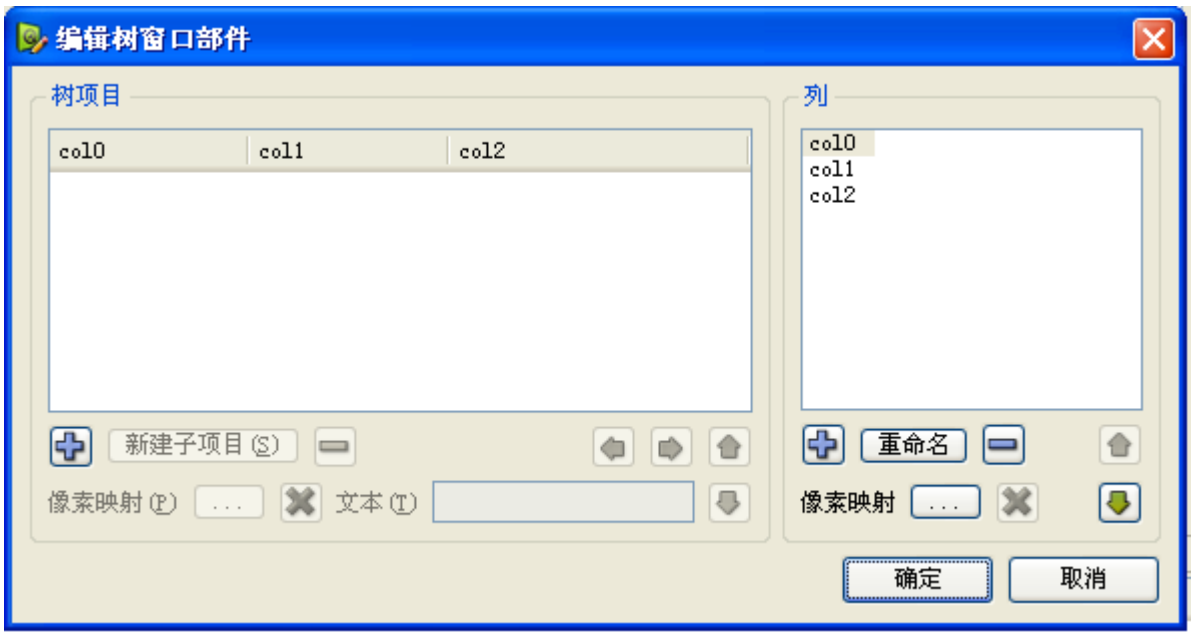
注解:

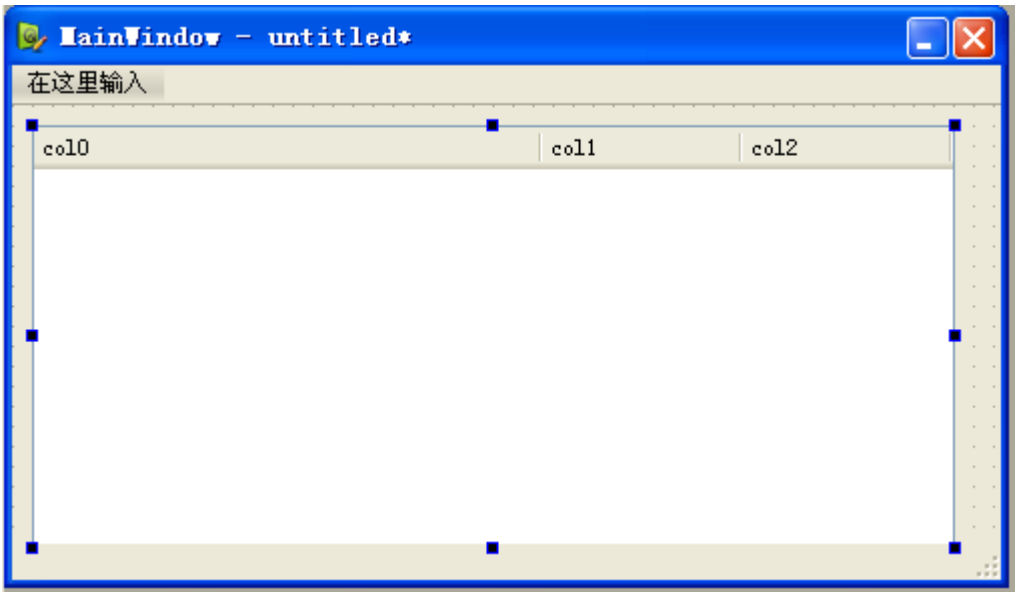
yolk的安装步骤请参考 [安装yolk]

在 PyQt4 中我们有三种列表部件可以使用：List View、Tree View 和 Table View。这些部件都可以在列表框架 中使用（Model and Item Based）。这里我们使用的是 Item Based：



在这个例子中，我们使用的是 Tree View。当你在 Tree View 部件上点击鼠标右键时，会出现“编辑项目”选项，通过它可以给部件增加列。需要注意的是列从 0 开始编号：





Tree View 给我们提供了丰富的功能，具体细节请参考文档。

QYolk

Yolk 可以通过 easy_install 命令安装：

```
easy_install yolk
```

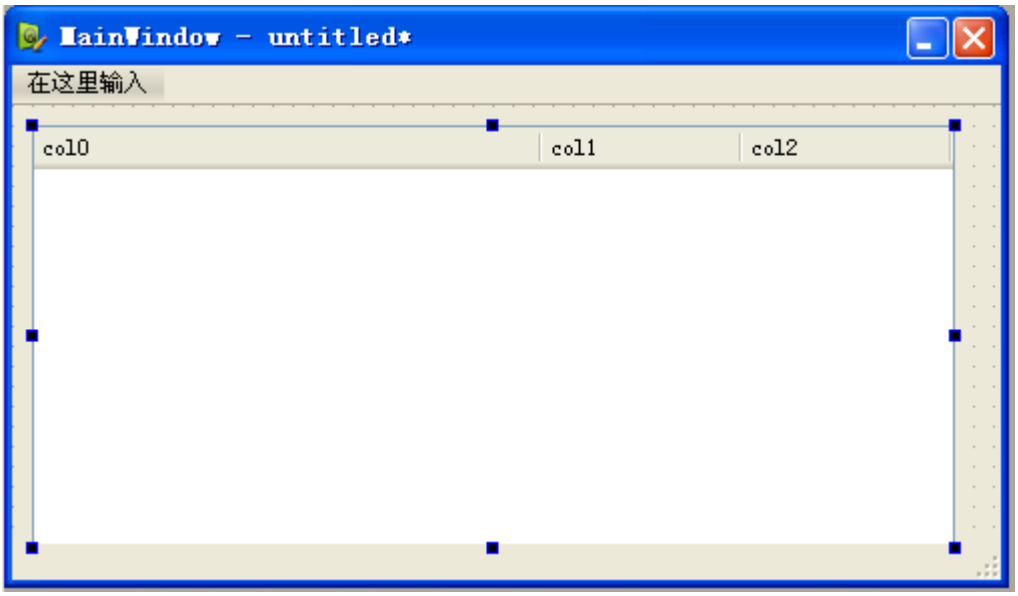
运行命令：

```
yolk -l
```

可以看到已经安装的包。我们将要把这些包列表通过 QtreeWidget 显示出来。数据通过以下方式获取：

```
from yolk import yolklib
packages = yolklib.Distributions()
for pkg in packages.get_distributions('all'):
       print pkg[0]
       print pkg[1]
       print '#####'
```

函数 get_distributions 返回包的信息：name + version，以及包的状态（Active/not-active）。我们把 列表部件命名为 treeList，把窗口命名为 QYolk，然后保存到 qyolk.ui。生成 qyolk.py 文件：



创建 start.py 文件：

```
# -*- coding: utf-8 -*-
import sys
from PyQt4 import QtCore, QtGui
from qyolk import Ui_QYolk
from yolk import yolklib

class StartQt4(QtGui.QMainWindow):
	def __init__(self, parent=None):
		QtGui.QWidget.__init__(self, parent)
		self.ui = Ui_QYolk()
		self.ui.setupUi(self)
		# set the widths of the columns
		self.ui.treeList.setColumnWidth(0,200)
		self.ui.treeList.setColumnWidth(1,100)

if __name__ == "__main__":
	app = QtGui.QApplication(sys.argv)
	myapp = StartQt4()
	myapp.show()
	sys.exit(app.exec_())
```

setColumnWidth 用于设置 QtreeView 中每列的宽度。0 对应第一列，1 对应第二列。下面我们利用 QtreeWidgetItem 向列表中添加数据：

```
a = QtGui.QtreeWidgetItem(self.ui.treeList)
a.setText(0, 'a')
a.setText(1, 'b')
a.setText(2, 'c')
```

QtreeWidgetItem 需要指定一个 QtreeWidget 对象，表示要添加数据的列表。方法 setText(Column ID, Text) 用来设置对应列的数据。我们通过一个循环来完成操作：

```
# -*- coding: utf-8 -*-
import sys
from PyQt4 import QtCore, QtGui
from qyolk import Ui_QYolk
from yolk import yolklib

class StartQt4(QtGui.QMainWindow):
	def __init__(self, parent=None):
		QtGui.QWidget.__init__(self, parent)
		self.ui = Ui_QYolk()
		self.ui.setupUi(self)
		# set the widths of the columns
		self.ui.treeList.setColumnWidth(0,200)
		self.ui.treeList.setColumnWidth(1,100)
		# generator which retuns list of installed packages
		packages = yolklib.Distributions()
		for pkg in packages.get_distributions('all'):
			a = QtGui.QtreeWidgetItem(self.ui.treeList)
			pk = str(pkg[0]).split(' ')
			if pkg[1]:
				status = 'Active'
			else:
				status = 'Not Active'
			a.setText(0, pk[0])
			a.setText(1, pk[1])
			a.setText(2, status)

if __name__ == "__main__":
	app = QtGui.QApplication(sys.argv)
	myapp = StartQt4()
	myapp.show()
```

```
	sys.exit(app.exec_())
```

列表数据已经读出来了。我们还可以根据包的活动状态设置不同的颜色：

```
# -*- coding: utf-8 -*-
import sys
from PyQt4 import QtCore, QtGui
from qyolk import Ui_QYolk
from yolk import yolklib

class StartQt4(QtGui.QMainWindow):
	def __init__(self, parent=None):
		QtGui.QWidget.__init__(self, parent)
		self.ui = Ui_QYolk()
		self.ui.setupUi(self)
		# set the widths of the columns
		self.ui.treeList.setColumnWidth(0,200)
		self.ui.treeList.setColumnWidth(1,100)
		# generator which retuns list of installed packages
		packages = yolklib.Distributions()
		for pkg in packages.get_distributions('all'):
			a = QtGui.QtreeWidgetItem(self.ui.treeList)
			pk = str(pkg[0]).split(' ')
			if pkg[1]:
				status = 'Active'
				a.setTextColor(0, QtGui.QColor(0, 0, 255))
				a.setTextColor(1, QtGui.QColor(0, 0, 255))
				a.setTextColor(2, QtGui.QColor(0, 0, 255))
			else:
				status = 'Not Active'
				a.setTextColor(0, QtGui.QColor(128, 128, 128))
				a.setTextColor(1, QtGui.QColor(128, 128, 128))
				a.setTextColor(2, QtGui.QColor(128, 128, 128))
			a.setText(0, pk[0])
			a.setText(1, pk[1])
			a.setText(2, status)

if __name__ == "__main__":
	app = QtGui.QApplication(sys.argv)
```

```
        myapp = StartQt4()
        myapp.show()
        sys.exit(app.exec_())
```

setTextColor(Column ID, QtGui.QColor(R, G, B))根据 RGB 来设置颜色。下面是运行效果：

安装yolk

注解:

该节为译者补充

由于我在自己的电脑上一直没有成功安装 easy_install，因此只好手工安装 yolk 了。 这里是在 Win-XP 下安装 yolk 步骤。

- 下载 setuptools-0.6c7.win32-py2.5.exe
- 下载 yolk-0.0.7.tar.gz

从本地下载：

- 下载 setuptools-0.6c7.win32-py2.5.exe
- 下载 yolk-0.0.7.tar.gz

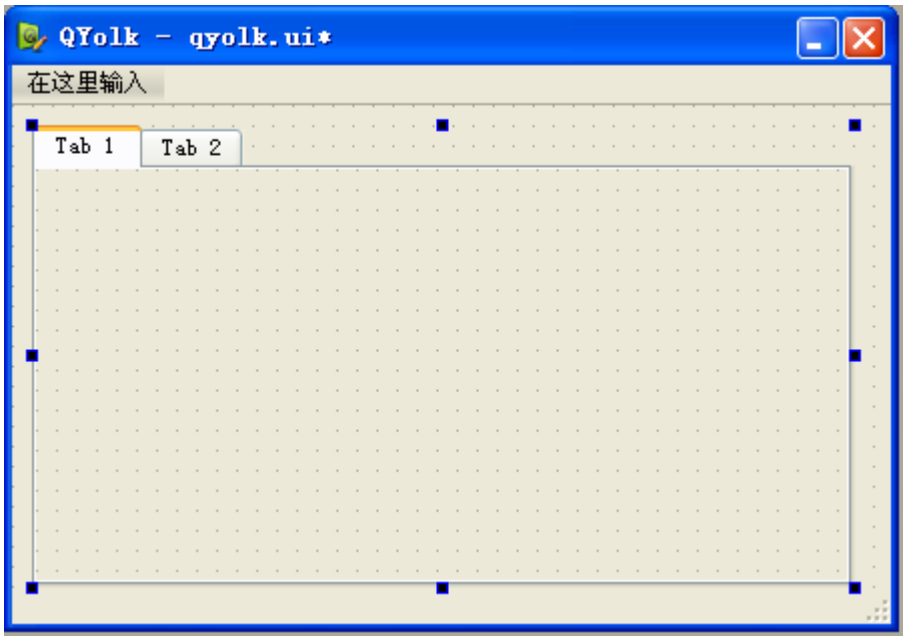
首先安装 setuptools，然后将 yolk-0.0.7 压缩包中的 yolk 子目录复制到 Python25-packages 中。现在应该就可以使用了 :)

QYolk II - 容器部件

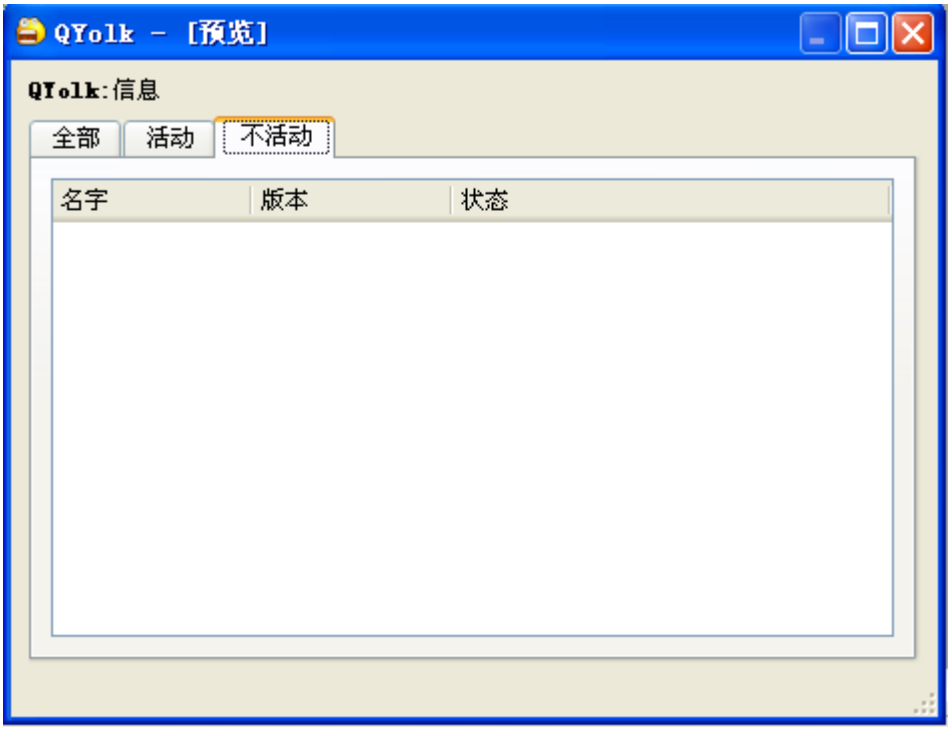
在这节教程中，我们将使用容器部件——Tab Widget。下图是上次教程实现的程序：

现在我们将使用 Tab Widget 在三个 tab 中分别显示 all packages/active/not active 包列表。

在 QtDesigner 中创建一个 Tab Widget，然后在第一个 tab 中创建一个 tree widget。当你把一个部件 放置到其他的容器部件中的时候，容器部件会高亮显示。如果你要编辑主窗口中的部件，主窗口也 会高亮显示。



通过右键菜单给 Tab Widget 添加 tab 页。然后在属性编辑器中修改每个标签页显示的文本。并且从第一个 tab 中 复制 tree widget 到后两个 tab 页中。



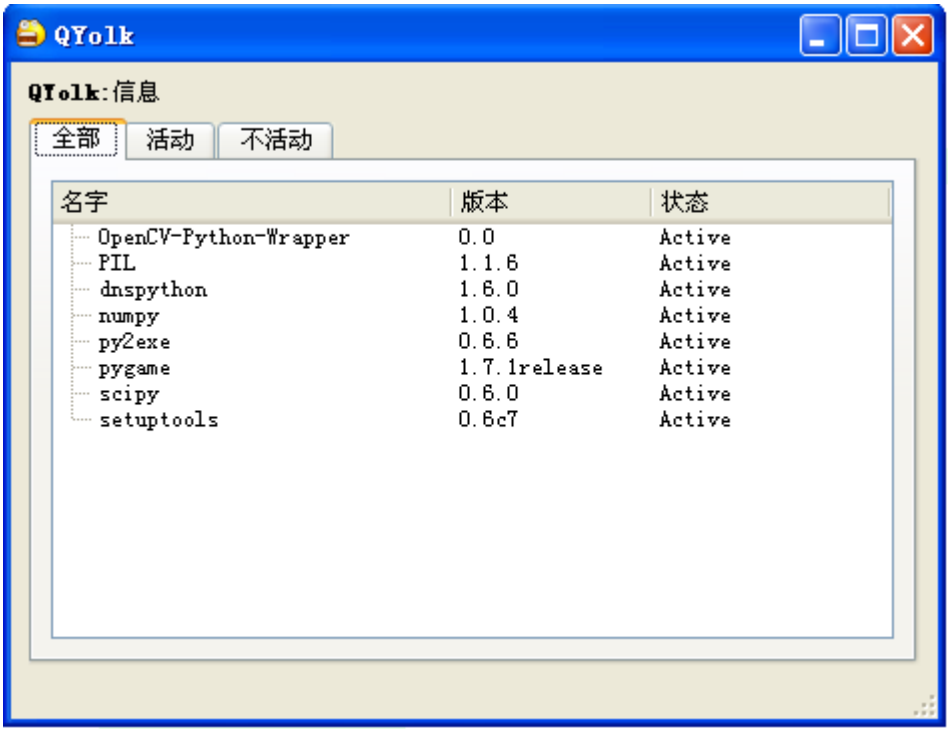
QtabWidget 重新命名为"pkgTabs"。三个 tree widget 分别命名为 "allList"/"activeList"/"notActiveList"。 QLabel 部件重新命名为"infoLabel"（目前还没有用到）。然后修改 start.py：

```
# -*- coding: utf-8 -*-
import sys
from PyQt4 import QtCore, QtGui
from qyolk import Ui_QYolk
from yolk import yolklib

class StartQt4(QtGui.QMainWindow):
       def __init__(self, parent=None):
              QtGui.QWidget.__init__(self, parent)
              self.ui = Ui_QYolk()
              self.ui.setupUi(self)
              # set the widths of the columns
              ################
              # All packages
              ################
              self.ui.allList.setColumnWidth(0,200)
              self.ui.allList.setColumnWidth(1,100)
              # generator which retuns list of installed packages
              packages = yolklib.Distributions()
              for pkg in packages.get_distributions('all'):
```

```
                a = QtGui.QtreeWidgetItem(self.ui.allList)
                pk = str(pkg[0]).split(' ')
                if pkg[1]:
                        status = 'Active'
                else:
                        status = 'Not Active'
                        a.setTextColor(0, QtGui.QColor(128, 128, 128))
                        a.setTextColor(1, QtGui.QColor(128, 128, 128))
                        a.setTextColor(2, QtGui.QColor(128, 128, 128))
                a.setText(0, pk[0])
                a.setText(1, pk[1])
                a.setText(2, status)
        ################
        # Active Packages
        ################
        # set the widths of the columns
        self.ui.activeList.setColumnWidth(0,200)
        self.ui.activeList.setColumnWidth(1,100)
        # generator which retuns list of active packages
        for pkg in packages.get_distributions('active'):
                a = QtGui.QtreeWidgetItem(self.ui.activeList)
                pk = str(pkg[0]).split(' ')
                a.setText(0, pk[0])
                a.setText(1, pk[1])
                a.setText(2, 'Active')
        ################
        # Not Active Packages
        ################
        # set the widths of the columns
        self.ui.notActiveList.setColumnWidth(0,200)
        self.ui.notActiveList.setColumnWidth(1,100)
        # generator which retuns list of not active packages
        for pkg in packages.get_distributions('nonactive'):
                a = QtGui.QtreeWidgetItem(self.ui.notActiveList)
                pk = str(pkg[0]).split(' ')
                a.setText(0, pk[0])
                a.setText(1, pk[1])
                a.setText(2, 'Not Active')
```

```
if __name__ == "__main__":
       app = QtGui.QApplication(sys.argv)
       myapp = StartQt4()
       myapp.show()
       sys.exit(app.exec_())
```

运行结果：



有参数的Singals/Slots

在切换 QtabWidget 的 tab 页的时候，会发射 currentChanged 信号：

```
void currentChanged (int)
```

该信号有一个 int 类型的参数。之前我们使用的信号都是没有参数的。在这里，int 参数对应 tab 页的编号， 并且每个 tab 页从 0 开始顺序编号。我们在连接 Singals/Slots 的时候，给 Slots 函数增加一个参数：

```
QtCore.QObject.connect(self.ui.pkgTabs,QtCore.SIGNAL("currentChanged(int)"),
self.tab_change)
def tab_change(self, tab_id):
       print tab_id
```

tab_id 是我们增加的一个参数，表示切换 tab 页的编号。signal 的参数和 slot 的参数依次对应。这样我们 就可以在切换 tab 页的时候，同步更新提示信息：

```
# -*- coding: utf-8 -*-
import sys
from PyQt4 import QtCore, QtGui
from qyolk import Ui_QYolk
from yolk import yolklib

class StartQt4(QtGui.QMainWindow):
       def __init__(self, parent=None):
              QtGui.QWidget.__init__(self, parent)
              self.ui = Ui_QYolk()
              self.ui.setupUi(self)
              # set the widths of the columns
              ################
              # All packages
              ################
              self.ui.allList.setColumnWidth(0,200)
              self.ui.allList.setColumnWidth(1,100)
              # generator which retuns list of installed packages
              packages = yolklib.Distributions()
              for pkg in packages.get_distributions('all'):
                     a = QtGui.QtreeWidgetItem(self.ui.allList)
                     pk = str(pkg[0]).split(' ')
                     if pkg[1]:
                            status = 'Active'
                     else:
                            status = 'Not Active'
                            a.setTextColor(0, QtGui.QColor(128, 128, 128))
                            a.setTextColor(1, QtGui.QColor(128, 128, 128))
                            a.setTextColor(2, QtGui.QColor(128, 128, 128))
                     a.setText(0, pk[0])
                     a.setText(1, pk[1])
                     a.setText(2, status)
              ################
              # Active Packages
              ################
              # set the widths of the columns
              self.ui.activeList.setColumnWidth(0,200)
              self.ui.activeList.setColumnWidth(1,100)
              # generator which retuns list of active packages
```

```
            for pkg in packages.get_distributions('active'):
                  a = QtGui.QtreeWidgetItem(self.ui.activeList)
                  pk = str(pkg[0]).split(' ')
                  a.setText(0, pk[0])
                  a.setText(1, pk[1])
                  a.setText(2, 'Active')
            ################
            # Not Active Packages
            ################
            # set the widths of the columns
            self.ui.notActiveList.setColumnWidth(0,200)
            self.ui.notActiveList.setColumnWidth(1,100)
            # generator which retuns list of not-active packages
            for pkg in packages.get_distributions('nonactive'):
                  a = QtGui.QtreeWidgetItem(self.ui.notActiveList)
                  pk = str(pkg[0]).split(' ')
                  a.setText(0, pk[0])
                  a.setText(1, pk[1])
                  a.setText(2, 'Not Active')

            # Signals

QtCore.QObject.connect(self.ui.pkgTabs,QtCore.SIGNAL("currentChanged(int)"),
self.tab_change)
      def tab_change(self, tab_id):
            if tab_id == 0:
                  self.ui.infoLabel.setText('<b>QYolk</b>: Browsing all
installed cheeseshop packages')
            elif tab_id == 1:
                  self.ui.infoLabel.setText('<b>QYolk</b>: Browsing active
packages')
            elif tab_id == 2:
                  self.ui.infoLabel.setText('<b>QYolk</b>: Browsing not
active packages (older versions)')

if __name__ == "__main__":
      app = QtGui.QApplication(sys.argv)
      myapp = StartQt4()
      myapp.show()
```

```
        sys.exit(app.exec_())
```

这里我们使用 self.ui.infoLabel.setText 来修改提示信息。下个 QYolk 程序我们将实现更多的功能。

PyQt4 文本编辑器 - 最终版

我们先描述新的特性：1. 使用 QFileSystemWatcher 来提示文件是否被外部改变；2. 创建并保存保存新的文件。 下面是 start.py 文件：

```
# -*- coding: utf-8 -*-
import sys
from PyQt4 import QtCore, QtGui
from edytor import Ui_notepad
import codecs
import codecs
from os.path import isfile

class StartQt4(QtGui.QMainWindow):
	def __init__(self, parent=None):
		QtGui.QWidget.__init__(self, parent)
		self.ui = Ui_notepad()
		self.ui.setupUi(self)
		self.watcher = QtCore.QFileSystemWatcher(self)

QtCore.QObject.connect(self.ui.button_open,QtCore.SIGNAL("clicked()"),
self.file_dialog)

QtCore.QObject.connect(self.ui.button_save,QtCore.SIGNAL("clicked()"),
self.file_save)

QtCore.QObject.connect(self.ui.editor_window,QtCore.SIGNAL("textChanged()"),
self.enable_save)

QtCore.QObject.connect(self.watcher,QtCore.SIGNAL("fileChanged(const
QString&)"), self.file_changed)
		self.filename = False
	def file_dialog(self):
		response = False
		# buttons texts
		SAVE = 'Save'
		DISCARD = 'Discard Changes'
		CANCEL = 'Cancel'
		# if we have changes then ask about them
		if self.ui.button_save.isEnabled():
```

```
                    message = QtGui.QMessageBox(self)
                    message.setText("Changes haven't been saved")
                    message.setWindowTitle('Notepad')
                    message.setIcon(QtGui.QMessageBox.Question)
                    message.addButton(SAVE, QtGui.QMessageBox.AcceptRole)
                    message.addButton(DISCARD,
QtGui.QMessageBox.DestructiveRole)
                    message.addButton(CANCEL, QtGui.QMessageBox.RejectRole)
                    message.setDetailedText('Unsaved Changes in: ' +
str(self.filename))
                    message.exec_()
                    response = message.clickedButton().text()
                    # save  file
                    if response == SAVE:
                           self.file_save()
                           self.ui.button_save.setEnabled(False)
                    # discard changes
                    elif response == DISCARD:
                           self.ui.button_save.setEnabled(False)
             # if we didn't canceled show the file dialog
             if response != CANCEL:
                    fd = QtGui.QFileDialog(self)
                    # remove old file from watcher
                    if self.filename:
                           self.watcher.removePath(self.filename)
                    self.filename = fd.getOpenFileName()
                    if isfile(self.filename):
                           s = codecs.open(self.filename,'r','utf-8').read()
                           self.ui.editor_window.setPlainText(s)
                           self.ui.button_save.setEnabled(False)
                           # add file to watcher
                           self.watcher.addPath(self.filename)
      def enable_save(self):
             self.ui.button_save.setEnabled(True)

      def file_changed(self, path):
             response = False
             # buttons texts
             SAVE = 'Save As'
```

```
            RELOAD = 'Reload File'
            CANCEL = 'Cancel'
            message = QtGui.QMessageBox(self)
            message.setText('Open file have been changed !')
            message.setWindowTitle('Notepad')
            message.setIcon(QtGui.QMessageBox.Warning)
            message.addButton(SAVE, QtGui.QMessageBox.AcceptRole)
            message.addButton(RELOAD, QtGui.QMessageBox.DestructiveRole)
            message.addButton(CANCEL, QtGui.QMessageBox.RejectRole)
            message.setDetailedText('The file "' + str(path) + '" have been
changed or removed by other application. What do you want to do ?')
            message.exec_()
            response = message.clickedButton().text()
            # save current file under a new or old name
            if response == SAVE:
                  fd = QtGui.QFileDialog(self)
                  newfile = fd.getSaveFileName()
                  if newfile:
                        s = codecs.open(newfile,'w','utf-8')

s.write(unicode(self.ui.editor_window.toPlainText()))
                        s.close()
                        self.ui.button_save.setEnabled(False)
                        # new file, remove old and add the new one to the watcher
                        if self.filename and str(newfile) !=
str(self.filename):
                              self.watcher.removePath(self.filename)
                              self.watcher.addPath(newfile)
                              self.filename = newfile
            # reload the text in the editor
            elif response == RELOAD:
                  s = codecs.open(self.filename,'r','utf-8').read()
                  self.ui.editor_window.setPlainText(s)
                  self.ui.button_save.setEnabled(False)

      def file_save(self):
            # save changes to existing file
            if self.filename and isfile(self.filename):
                  # don't react on our changes
```

```
                    self.watcher.removePath(self.filename)
                    s = codecs.open(self.filename,'w','utf-8')
                    s.write(unicode(self.ui.editor_window.toPlainText()))
                    s.close()
                    self.ui.button_save.setEnabled(False)
                    self.watcher.addPath(self.filename)
             # save a new file
             else:
                    fd = QtGui.QFileDialog(self)
                    newfile = fd.getSaveFileName()
                    if newfile:
                           s = codecs.open(newfile,'w','utf-8')

s.write(unicode(self.ui.editor_window.toPlainText()))
                           s.close()
                           self.ui.button_save.setEnabled(False)

if __name__ == "__main__":
       app = QtGui.QApplication(sys.argv)
       myapp = StartQt4()
       myapp.show()
       sys.exit(app.exec_())
```

函数 file_changed(self, path)在文件被外部程序修改的时候，由 QFileSystemWatcher 的 fileChanged(const QString&)信号引发：

```
       def file_changed(self, path):
              response = False
              # buttons texts
              SAVE = 'Save As'
              RELOAD = 'Reload File'
              CANCEL = 'Cancel'
              message = QtGui.QMessageBox(self)
              message.setText('Open file have been changed !')
              message.setWindowTitle('Notepad')
              message.setIcon(QtGui.QMessageBox.Warning)
              message.addButton(SAVE, QtGui.QMessageBox.AcceptRole)
              message.addButton(RELOAD, QtGui.QMessageBox.DestructiveRole)
              message.addButton(CANCEL, QtGui.QMessageBox.RejectRole)
```

```
            message.setDetailedText('The file "' + str(path) + '" have been
changed or removed by other application. What do you want to do ?')
            message.exec_()
            response = message.clickedButton().text()
            # save current file under a new or old name
            if response == SAVE:
                  fd = QtGui.QFileDialog(self)
                  newfile = fd.getSaveFileName()
                  if newfile:
                        s = codecs.open(newfile,'w','utf-8')

s.write(unicode(self.ui.editor_window.toPlainText()))
                        s.close()
                        self.ui.button_save.setEnabled(False)
                        # new file, remove old and add the new one to the watcher
                        if self.filename and str(newfile) !=
str(self.filename):
                              self.watcher.removePath(self.filename)
                              self.watcher.addPath(newfile)
                              self.filename = newfile
            # reload the text in the editor
            elif response == RELOAD:
                  s = codecs.open(self.filename,'r','utf-8').read()
                  self.ui.editor_window.setPlainText(s)
                  self.ui.button_save.setEnabled(False)
```

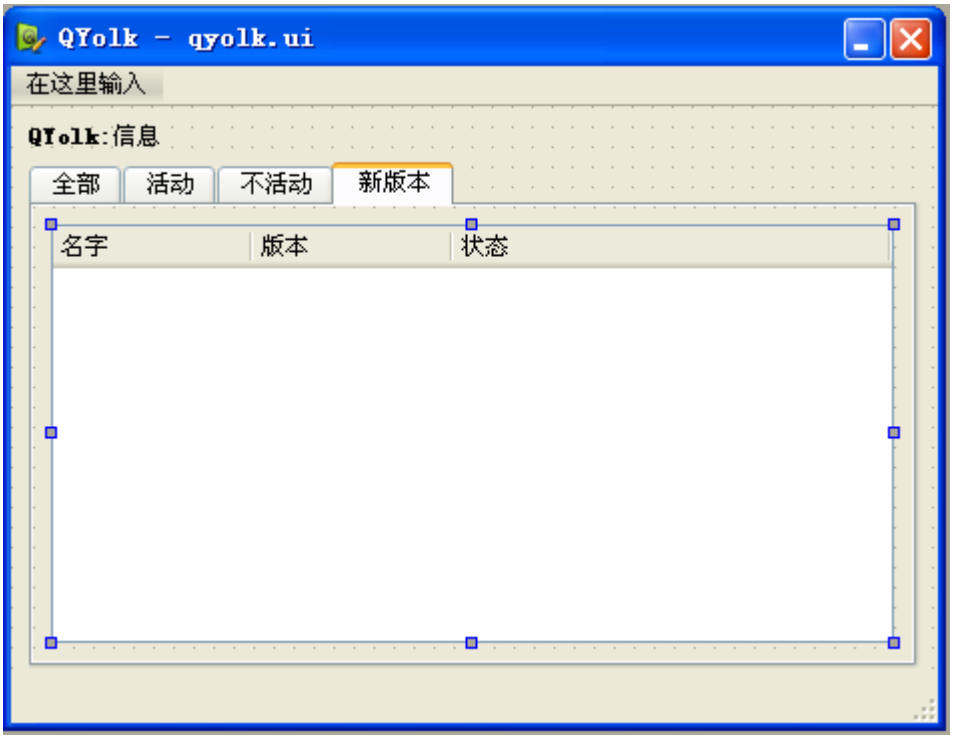
通过 QMessageBox 让用户选择，是重新装载文件，还是将当前文件作为一个新文件重新保存。保存文件名通过 QFileDialog 的 getSaveFileName 获取。在使用 QFileSystemWatcher 的时候，我们需要通过 adding/removing 来添加/删除要观察的路径列表。

QYolk III - 升级包列表

在这个练习中，我们将增加一个 tab 页，用于显示- 升级包列表。通过下面代码获取列表：

```
from yolk.cli import get_pkglist
from yolk.yolklib import get_highest_version, Distributions
from yolk.pypi import CheeseShop
import pkg_resources
def get_fresh_updates(package_name="", version=""):
       ret = []
       pypi = CheeseShop()
       dists = Distributions()
       for pkg in get_pkglist():
              for (dist, active) in dists.get_distributions("all", pkg,
dists.get_highest_installed(pkg)):
                     (project_name, versions) =
pypi.query_versions_pypi(dist.project_name, True)
                     if versions:
                            newest = get_highest_version(versions)
                            if newest != dist.version:
                                   if
pkg_resources.parse_version(dist.version) <
pkg_resources.parse_version(newest):
                                          ret.append([project_name,
dist.version, newest])
       return ret
print get_fresh_updates()
```

在检测包是否有新版本的时候，运行速度可能有些慢。在后面我们会设计一个缓冲来处理这个问题。先给 QtreeWidget 添加一个 tab 页，里面包含一个名字为"updatesList"的 tree list。



更新 start.py：

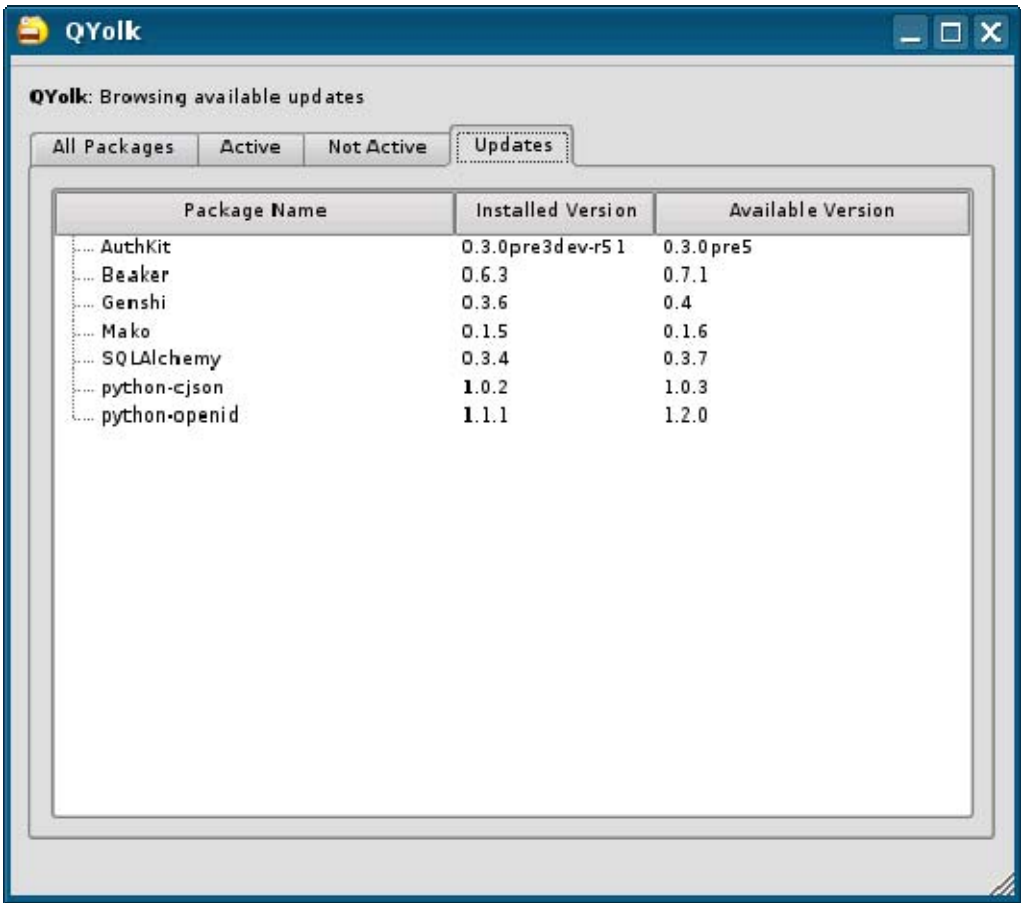
```
# -*- coding: utf-8 -*-
import sys
from PyQt4 import QtCore, QtGui
from qyolk import Ui_QYolk
from yolk import yolklib

class StartQt4(QtGui.QMainWindow):
       def __init__(self, parent=None):
              QtGui.QWidget.__init__(self, parent)
              self.ui = Ui_QYolk()
              self.ui.setupUi(self)
              # set the widths of the columns
              ################
              # All packages
              ################
              self.ui.allList.setColumnWidth(0,200)
              self.ui.allList.setColumnWidth(1,100)
              # generator which retuns list of installed packages
              packages = yolklib.Distributions()
              for pkg in packages.get_distributions('all'):
                     a = QtGui.QtreeWidgetItem(self.ui.allList)
```

```
                pk = str(pkg[0]).split(' ')
                if pkg[1]:
                        status = 'Active'
                else:
                        status = 'Not Active'
                        a.setTextColor(0, QtGui.QColor(128, 128, 128))
                        a.setTextColor(1, QtGui.QColor(128, 128, 128))
                        a.setTextColor(2, QtGui.QColor(128, 128, 128))
                a.setText(0, pk[0])
                a.setText(1, pk[1])
                a.setText(2, status)
        ################
        # Active Packages
        ################
        # set the widths of the columns
        self.ui.activeList.setColumnWidth(0,200)
        self.ui.activeList.setColumnWidth(1,100)
        # generator which retuns list of active packages
        for pkg in packages.get_distributions('active'):
                a = QtGui.QtreeWidgetItem(self.ui.activeList)
                pk = str(pkg[0]).split(' ')
                a.setText(0, pk[0])
                a.setText(1, pk[1])
                a.setText(2, 'Active')
        ################
        # Not Active Packages
        ################
        # set the widths of the columns
        self.ui.notActiveList.setColumnWidth(0,200)
        self.ui.notActiveList.setColumnWidth(1,100)
        # generator which retuns list of not-active packages
        for pkg in packages.get_distributions('nonactive'):
                a = QtGui.QtreeWidgetItem(self.ui.notActiveList)
                pk = str(pkg[0]).split(' ')
                a.setText(0, pk[0])
                a.setText(1, pk[1])
                a.setText(2, 'Not Active')

        # Signals
```

```
QtCore.QObject.connect(self.ui.pkgTabs,QtCore.SIGNAL("currentChanged(int)"),
self.tab_change)
	def tab_change(self, tab_id):
		if tab_id == 0:
			self.ui.infoLabel.setText('<b>QYolk</b>: Browsing all
installed cheeseshop packages')
		elif tab_id == 1:
			self.ui.infoLabel.setText('<b>QYolk</b>: Browsing active
packages')
		elif tab_id == 2:
			self.ui.infoLabel.setText('<b>QYolk</b>: Browsing not
active packages (older versions)')
		elif tab_id == 3:
			self.ui.infoLabel.setText('<b>QYolk</b>: Browsing
available updates')
			################
			# List Updates
			################
			# set the widths of the columns
			self.ui.updatesList.setColumnWidth(0,200)
			self.ui.updatesList.setColumnWidth(1,150)
			for pkg in get_fresh_updates():
				a = QtGui.QtreeWidgetItem(self.ui.updatesList)
				a.setText(0, pkg[0])
				a.setText(1, pkg[1])
				a.setText(2, pkg[2])

from yolk.cli import get_pkglist
from yolk.yolklib import get_highest_version, Distributions
from yolk.pypi import CheeseShop
import pkg_resources
def get_fresh_updates(package_name="", version=""):
	ret = []
	pypi = CheeseShop()
	dists = Distributions()
	for pkg in get_pkglist():
		for (dist, active) in dists.get_distributions("all", pkg,
dists.get_highest_installed(pkg)):
```

```
                    (project_name, versions) =
pypi.query_versions_pypi(dist.project_name, True)
                    if versions:
                           newest = get_highest_version(versions)
                           if newest != dist.version:
                                  if
pkg_resources.parse_version(dist.version) <
pkg_resources.parse_version(newest):
                                         ret.append([project_name,
dist.version, newest])
       return ret

if __name__ == "__main__":
       app = QtGui.QApplication(sys.argv)
       myapp = StartQt4()
       myapp.show()
       sys.exit(app.exec_())
```

译者：这里我没有运行成功，因此用的是原网站的截图。



新改动是在 get_fresh_updates 函数中，增加了新 tab 页的处理：

```
            elif tab_id == 3:
                  self.ui.infoLabel.setText('<b>QYolk</b>: Browsing
available updates')
                  ################
                  # List Updates
                  ################
                  # set the widths of the columns
                  self.ui.updatesList.setColumnWidth(0,200)
                  self.ui.updatesList.setColumnWidth(1,150)
                  for pkg in get_fresh_updates():
                        a = QtGui.QtreeWidgetItem(self.ui.updatesList)
                        a.setText(0, pkg[0])
                        a.setText(1, pkg[1])
                        a.setText(2, pkg[2])
```

当切换到"Updates" tab 的时候，可能会花一些时间（依赖网络环境）。我们使用 Pickle 来缓冲信息：

```
from os.path import expanduser
import cPickle
userpath = expanduser('~')
f = open(userpath + '/test', 'w')
cPickle.dump(['a', 'b', 'c'], f)
```

读 pickled 数据：

```
f = open(userpath + '/test', 'r')
print cPickle.load(f)
```

给 get_fresh_updates 增加缓冲：

```
def get_fresh_updates(package_name="", version=""):
      userpath = expanduser('~')
      now = datetime.now()
      # do we have the cache
      if isfile(userpath + '/.qyolk'):
            f = open(userpath + '/.qyolk', 'r')
            cache = cPickle.load(f)
            check_time = now - timedelta(hours=24)
            if cache[0] > check_time:
```

```
                    # fresh cache, use it
                    return cache[1]

        # no cache, get updates and create the cace
        ret = []
        pypi = CheeseShop()
        dists = Distributions()
        for pkg in get_pkglist():
                for (dist, active) in dists.get_distributions("all", pkg,
dists.get_highest_installed(pkg)):
                    (project_name, versions) =
pypi.query_versions_pypi(dist.project_name, True)
                    if versions:
                            newest = get_highest_version(versions)
                            if newest != dist.version:
                                    if
pkg_resources.parse_version(dist.version) <
pkg_resources.parse_version(newest):
                                            ret.append([project_name,
dist.version, newest])
        f = open(userpath + '/.qyolk', 'w')
        cPickle.dump([now, ret], f)
        return ret
```

cPickle 保存一个列表[Date, list of upgrades]。我们检测缓冲文件是否存在，如果存在检测文件是否大于 24 小时没有更新。 如果文件不存在，我们将创建一个新的缓冲文件。下面是 start.py：

```
# -*- coding: utf-8 -*-
import sys
from PyQt4 import QtCore, QtGui
from qyolk import Ui_QYolk
from yolk import yolklib
from os.path import expanduser
import cPickle
from yolk.cli import get_pkglist
from yolk.yolklib import get_highest_version, Distributions
from yolk.pypi import CheeseShop
import pkg_resources
from os.path import isfile
from datetime import timedelta
```

```
from datetime import datetime

class StartQt4(QtGui.QMainWindow):
	def __init__(self, parent=None):
		QtGui.QWidget.__init__(self, parent)
		self.ui = Ui_QYolk()
		self.ui.setupUi(self)

		#splash = QtGui.QSplashScreen(self)
		#splash.drawContents()

		################
		# All packages
		################
		self.ui.allList.setColumnWidth(0,200)
		self.ui.allList.setColumnWidth(1,100)
		# generator which retuns list of installed packages
		packages = yolklib.Distributions()
		for pkg in packages.get_distributions('all'):
			a = QtGui.QtreeWidgetItem(self.ui.allList)
			pk = str(pkg[0]).split(' ')
			if pkg[1]:
				status = 'Active'
			else:
				status = 'Not Active'
				a.setTextColor(0, QtGui.QColor(128, 128, 128))
				a.setTextColor(1, QtGui.QColor(128, 128, 128))
				a.setTextColor(2, QtGui.QColor(128, 128, 128))
			a.setText(0, pk[0])
			a.setText(1, pk[1])
			a.setText(2, status)
		################
		# Active Packages
		################
		# set the widths of the columns
		self.ui.activeList.setColumnWidth(0,200)
		self.ui.activeList.setColumnWidth(1,100)
		# generator which retuns list of active packages
		for pkg in packages.get_distributions('active'):
```

```
                a = QtGui.QtreeWidgetItem(self.ui.activeList)
                pk = str(pkg[0]).split(' ')
                a.setText(0, pk[0])
                a.setText(1, pk[1])
                a.setText(2, 'Active')
        ################
        # Not Active Packages
        ################
        # set the widths of the columns
        self.ui.notActiveList.setColumnWidth(0,200)
        self.ui.notActiveList.setColumnWidth(1,100)
        # generator which retuns list of not-active packages
        for pkg in packages.get_distributions('nonactive'):
                a = QtGui.QtreeWidgetItem(self.ui.notActiveList)
                pk = str(pkg[0]).split(' ')
                a.setText(0, pk[0])
                a.setText(1, pk[1])
                a.setText(2, 'Not Active')
        ################
        # List Updates
        ################
        # set the widths of the columns
        self.ui.updatesList.setColumnWidth(0,200)
        self.ui.updatesList.setColumnWidth(1,150)
        for pkg in get_fresh_updates():
                a = QtGui.QtreeWidgetItem(self.ui.updatesList)
                a.setText(0, pkg[0])
                a.setText(1, pkg[1])
                a.setText(2, pkg[2])

        # Signals

QtCore.QObject.connect(self.ui.pkgTabs,QtCore.SIGNAL("currentChanged(int)"),
self.tab_change)
    def tab_change(self, tab_id):
        if tab_id == 0:
                self.ui.infoLabel.setText('<b>QYolk</b>: Browsing all
installed cheeseshop packages')
        elif tab_id == 1:
```

```
                    self.ui.infoLabel.setText('<b>QYolk</b>: Browsing active
packages')
             elif tab_id == 2:
                    self.ui.infoLabel.setText('<b>QYolk</b>: Browsing not
active packages (older versions)')
             elif tab_id == 3:
                    self.ui.infoLabel.setText('<b>QYolk</b>: Browsing
available updates')

def get_fresh_updates(package_name="", version=""):
       userpath = expanduser('~')
       now = datetime.now()
       # do we have the cache
       if isfile(userpath + '/.qyolk'):
              f = open(userpath + '/.qyolk', 'r')
              cache = cPickle.load(f)
              check_time = now - timedelta(hours=24)
              if cache[0] > check_time:
                     # fresh cache, use it
                     return cache[1]

       # no cache, get updates and create the cace
       ret = []
       pypi = CheeseShop()
       dists = Distributions()
       for pkg in get_pkglist():
              for (dist, active) in dists.get_distributions("all", pkg,
dists.get_highest_installed(pkg)):
                     (project_name, versions) =
pypi.query_versions_pypi(dist.project_name, True)
                     if versions:
                            newest = get_highest_version(versions)
                            if newest != dist.version:
                                   if
pkg_resources.parse_version(dist.version) <
pkg_resources.parse_version(newest):
                                          ret.append([project_name,
dist.version, newest])
       f = open(userpath + '/.qyolk', 'w')
       cPickle.dump([now, ret], f)
```

```
        return ret


if __name__ == "__main__":
        app = QtGui.QApplication(sys.argv)
        myapp = StartQt4()
        myapp.show()
        sys.exit(app.exec_())
```

更新包的信息我们移动到了__init__ —— 它将在程序启动的时候运行。如果缓冲文件不存在的话，窗口 将在成功更新包信息之后被创建（这中间可能要花一些时间）。如果缓冲文件已经则不会有太大的问题。 在下一次的教程中，我们将用 QSplashScreen 给程序增加一个启动画面，这样在窗口被创建的时候可以 给用户提供一些有用的信息。

更多内容，请参阅：

http://www.rkblog.rk.edu.pl/w/p/introduction-pyqt4/