★ QMarkdowner

QMarkdowner

1. Introduction（简介）

```
    An editor for Markdown based on PyQt4 with Metro style;You can use this editor
to markdown what you want to markdown.
    基于PyQt4的markdown编辑器，采用Metro style
```

2. Feature（特性）

```
    You can write your markdown worlds on the left side, also you can priview
what you wirte in html style;
    If you  want to preview in an big windown, you can click Markdown预览（Preview）
button.
    支持边编辑边预览和全屏预览；
    支持导出编辑的markdown文件；
    支持多风格预览和导出：四种主流的markdown预览主题可供选择，导出的html中包含相应的风格，无须额外css支持；
```

3. Improvement(改进)

> 利用pyqt4原生的控件实现整个逻辑
>
> 功能增加改进
>
> > 一键发布到Evernote, github， blog等
> > 支持markdown到PDF的转换和导出

4. Greet(致敬)

```
感谢：   Chrome插件MaDe作者Lyric Wai，
         ulipad和uliweb作者limodou，
         Good朋友jack_zh，
         PyQt and PySide群里的樱桃大丸子等大神.
联系邮箱：dragondjf@gmail.com, ding465398889@163.com
联系QQ：465398889
GitHub地址：https://github.com/dragondjf/
软件发布地址：http://aj5sjte66c.l35.yunpan.cn/lk/QXW2ntCvGw83j
刻骨铭心：感谢开源精神，分享成就自我！
```

5. 代码预览

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import os
import logging
from logging.handlers import RotatingFileHandler
from PyQt4 import QtGui
from PyQt4 import QtCore
import json
import time


#主日志保存在log/QSoftkeyer.log
logging.root.setLevel(logging.INFO)
logging.root.propagate = 0
loghandler = RotatingFileHandler(os.path.join("log", "QMain.log"), maxBytes=10 * 1024 * 1024, backupCount=100)
loghandler.setFormatter(logging.Formatter('%(asctime)s %(levelname)8s [%(filename)16s:%(lineno)04s] %(message)s'))
loghandler.level = logging.INFO
logging.root.addHandler(loghandler)
logger = logging.root
logger.propagate = 0


from config import windowsoptions
import config
```

```
from effects import *
from childpages import *
from guiutil.utils import set_skin, set_bg
import utildialog


class MetroWindow(QtGui.QWidget):

    def __init__(self, parent=None):
        super(MetroWindow, self).__init__(parent)

        self.pagetags = windowsoptions['mainwindow']['centralwindow']['pagetags']
        self.pagetags_zh = windowsoptions['mainwindow']['centralwindow']['pagetags_zh']
        self.initUI()

    def initUI(self):
        self.pagecount = len(self.pagetags_zh)  # 页面个数
        # self.createNavigation()
        self.pages = QtGui.QStackedWidget()  # 创建堆控件

        # self.pages.addWidget(self.navigationPage)

        self.createChildPages()  # 创建子页面

        # self.createConnections()
        mainLayout = QtGui.QHBoxLayout()
        mainLayout.addWidget(self.pages)
        self.setLayout(mainLayout)
        self.layout().setContentsMargins(0, 0, 0, 0)

        self.faderWidget = None
        self.connect(self.pages, QtCore.SIGNAL("currentChanged(int)"), self.fadeInWidget)  # 页面切换时淡入淡出效果

    def createNavigation(self):
        '''
            创建导航页面
        '''
        self.navigationPage = NavigationPage()

    def createChildPages(self):
        '''
            创建子页面
        '''
        for buttons in self.pagetags:
            for item in buttons:
                page = item + 'Page'
                childpage = 'child' + page

                if hasattr(sys.modules[__name__], page):
                    setattr(self, page, getattr(sys.modules[__name__], page)(self))
                else:
                    setattr(self, page, getattr(sys.modules[__name__], 'BasePage')(self))
                setattr(self, childpage, ChildPage(self, getattr(self, page)))
                self.pages.addWidget(getattr(self, childpage))

    def createConnections(self):
        '''
            创建按钮与页面的链接
        '''
        for buttons in self.pagetags:
            for item in buttons:
                button = item + 'Button'
                getattr(self.navigationPage, button).clicked.connect(self.childpageChange)

    def childpageChange(self):
        '''
            页面切换响应函数
        '''
        currentpage = getattr(self, unicode('child' + self.sender().objectName()[:-6]) + 'Page')
        if hasattr(self, 'navigationPage'):
            if currentpage is self.navigationPage:
                currentpage.parent.parent().statusBar().hide()
        self.pages.setCurrentWidget(currentpage)
        self.sender().setFocus()

        #切换QChrome页面时进行实时刷新显示预览
        if isinstance(currentpage.child, QChromePage):
            currentpage.child.refreshcontent()

    @QtCore.pyqtSlot()
    def backnavigationPage(self):
        self.parent().statusBar().hide()
        self.pages.setCurrentWidget(self.navigationPage)

    @QtCore.pyqtSlot()
    def backPage(self):
        index = self.pages.currentIndex()
```

```
        if index == 1:
            self.parent().statusBar().hide()
            self.pages.setCurrentWidget(self.navigationPage)
        else:
            self.pages.setCurrentIndex(index - 1)

    @QtCore.pyqtSlot()
    def forwardnextPage(self):
        index = self.pages.currentIndex()
        if index < self.pagecount:
            self.pages.setCurrentIndex(index + 1)
        else:
            self.parent().statusBar().hide()
            self.pages.setCurrentWidget(self.navigationPage)

    def fadeInWidget(self, index):
        '''
            页面切换时槽函数实现淡入淡出效果
        '''
        self.faderWidget = FaderWidget(self.pages.widget(0.5))
        self.faderWidget.start()


class MainWindow(QtGui.QMainWindow):
    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)

        self.initFrame()
        self.centeralwindow = MetroWindow(self)
        self.setCentralWidget(self.centeralwindow)

        self.createMenus()
        self.createToolbars()
        self.createStatusbar()

        self.setskin()

        currentpage = self.centralWidget().pages.currentWidget()
        currentpage.navigation.setVisible(windowsoptions['mainwindow']['navigationvisual'])

    def initFrame(self):
        title = windowsoptions['mainwindow']['title']
        postion = windowsoptions['mainwindow']['postion']
        minsize = windowsoptions['mainwindow']['minsize']
        size = windowsoptions['mainwindow']['size']
        windowicon = windowsoptions['mainwindow']['windowicon']
        fullscreenflag = windowsoptions['mainwindow']['fullscreenflag']
        navigationvisual = windowsoptions['mainwindow']['navigationvisual']

        self.setWindowTitle(title)
        self.setWindowIcon(QtGui.QIcon(windowicon))  # 设置程序图标
        self.setMinimumSize(minsize[0], minsize[1])
        width = QtGui.QDesktopWidget().availableGeometry().width() * 5 / 6
        height = QtGui.QDesktopWidget().availableGeometry().height() * 7 / 8
        self.setGeometry(postion[0], postion[1], width, height)  # 初始化窗口位置和大小
        self.center()  # 将窗口固定在屏幕中间
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)

        self.fullscreenflag = fullscreenflag  # 初始化时非窗口最大话标志
        if self.fullscreenflag:
            self.showFullScreen()
        else:
            self.showNormal()

        self.navigationvisual = navigationvisual   # 导航标志，初始化时显示导航
        self.layout().setContentsMargins(0, 0, 0, 0)

        # self.setWindowFlags(QtCore.Qt.CustomizeWindowHint)  # 隐藏标题栏， 可以拖动边框改变大小
        # self.setWindowFlags(QtCore.Qt.FramelessWindowHint)  # 隐藏标题栏， 无法改变大小
        self.setWindowFlags(QtCore.Qt.FramelessWindowHint | \
        QtCore.Qt.WindowSystemMenuHint | QtCore.Qt.WindowMinimizeButtonHint)  # 无边框， 带系统菜单， 可以最小化

    def setskin(self):
        for buttons in windowsoptions['mainwindow']['centralwindow']['pagetags']:
            for item in buttons:
                childpage = getattr(self.centeralwindow, 'child' + item + 'Page')
                set_skin(childpage, os.sep.join(['skin', 'qss', 'MetroNavigationBar.qss']))    # 设置导航工具条的样式

        set_skin(self, os.sep.join(['skin', 'qss', 'MetroMainwindow.qss']))  # 设置主窗口样式

    def center(self):
        qr = self.frameGeometry()
        cp = QtGui.QDesktopWidget().availableGeometry().center()
        qr.moveCenter(cp)
        self.move(qr.topLeft())

    def createMenus(self):
```

```
        menusettings = windowsoptions['mainwindow']['menusettings']
        menubar = self.menuBar()
        menubar.setVisible(menusettings['visual'])
        for menu in menusettings['menus']:
            setattr(
                self,
                '%smenu' % menu['name'],
                menubar.addMenu(u'%s%s' % (menu['name'], menu['name_zh']))
            )
            submenu = getattr(self, '%smenu' % menu['name'])
            for menuaction in menu['actions']:
                setattr(
                    self,
                    '%sAction' % menuaction['trigger'],
                    QtGui.QAction(
                        QtGui.QIcon(QtGui.QPixmap(menuaction['icon'])),
                        '%s%s' % (menuaction['name'], menuaction['name_zh']),
                        self
                    )
                )
                if hasattr(self, 'action%s' % menuaction['trigger']):
                    action = getattr(self, '%sAction' % menuaction['trigger'])
                    action.setShortcut(QtGui.QKeySequence(menuaction['shortcut']))
                    submenu.addAction(action)
                    action.triggered.connect(
                        getattr(self, 'action%s' % menuaction['trigger'])
                    )
                else:
                    action = getattr(self, '%sAction' % menuaction['trigger'])
                    action.setShortcut(QtGui.QKeySequence(menuaction['shortcut']))
                    submenu.addAction(action)
                    action.triggered.connect(
                        getattr(self, 'actionNotImplement')
                    )

    def createToolbars(self):
        toolbarsettings = windowsoptions['mainwindow']['toolbarsettings']
        self.toolbar = QtGui.QToolBar(self)
        self.toolbar.setMovable(toolbarsettings['movable'])
        self.toolbar.setVisible(toolbarsettings['visual'])
        self.addToolBar(toolbarsettings['dockArea'], self.toolbar)

        for toolbar in toolbarsettings['toolbars']:
            setattr(
                self,
                '%sAction' % toolbar['trigger'],
                QtGui.QAction(
                    QtGui.QIcon(QtGui.QPixmap(toolbar['icon'])),
                    '%s%s' % (toolbar['name'], toolbar['name_zh']),
                    self
                )
            )
            if hasattr(self, 'action%s' % toolbar['trigger']):
                action = getattr(self, '%sAction' % toolbar['trigger'])
                action.setShortcut(QtGui.QKeySequence(toolbar['shortcut']))
                action.setToolTip(toolbar['tooltip'])
                self.toolbar.addAction(action)
                action.triggered.connect(
                    getattr(self, 'action%s' % toolbar['trigger'])
                )
                self.toolbar.widgetForAction(action).setObjectName(toolbar['id'])
            else:
                action = getattr(self, '%sAction' % toolbar['trigger'])
                action.setShortcut(QtGui.QKeySequence(toolbar['shortcut']))
                action.setToolTip(toolbar['tooltip'])
                self.toolbar.addAction(action)
                action.triggered.connect(
                    getattr(self, 'actionNotImplement')
                )
                self.toolbar.widgetForAction(action).setObjectName(toolbar['id'])

    def createStatusbar(self):
        statusbarsettings = windowsoptions['mainwindow']['statusbarsettings']
        self.statusbar = QtGui.QStatusBar()
        self.setStatusBar(self.statusbar)
        self.statusbar.showMessage(statusbarsettings['initmessage'])
        self.statusbar.setMinimumHeight(statusbarsettings['minimumHeight'])
        self.statusbar.setVisible(statusbarsettings['visual'])

    def actionAbout(self):
        pass

    def actionNotImplement(self):
        utildialog.msg(u'This action is not Implemented', windowsoptions['msgdialog'])

    @QtCore.pyqtSlot()
    def windowMaxNormal(self):
```

```
        if self.isFullScreen():
            self.showNormal()
            self.sender().setObjectName("MaxButton")
            set_skin(self, os.sep.join(['skin', 'qss', 'MetroMainwindow.qss']))  # 设置主窗口样式
        else:
            self.showFullScreen()
            self.sender().setObjectName("MaxNormalButton")
            set_skin(self, os.sep.join(['skin', 'qss', 'MetroMainwindow.qss']))  # 设置主窗口样式

    def closeEvent(self, evt):
        flag, exitflag = utildialog.exit(windowsoptions['exitdialog'])
        if flag:
            for item in exitflag:
                if item == 'minRadio' and exitflag[item]:
                    self.showMinimized()
                    evt.ignore()
                elif item == 'exitRadio' and exitflag[item]:
                    evt.accept()
                elif item == 'exitsaveRadio' and exitflag[item]:
                    evt.accept()
                    self.saveoptions()
                    with open(os.sep.join([os.getcwd(), 'options', 'windowsoptions.json']), 'wb') as f:
                        json.dump(windowsoptions, f, indent=1)
        else:
            evt.ignore()

    def saveoptions(self):
        windowsoptions['mainwindow']['fullscreenflag'] = self.fullscreenflag
        windowsoptions['mainwindow']['navigationvisual'] = \
            self.centeralwindow.pages.currentWidget().navigation.isVisible()
        windowsoptions['mainwindow']['menusettings']['visual'] = \
            self.menuBar().isVisible()
        windowsoptions['mainwindow']['statusbarsettings']['visual'] = \
            self.statusBar().isVisible()

    def keyPressEvent(self, evt):
        if evt.key() == QtCore.Qt.Key_Escape:
            self.close()
        elif evt.key() == QtCore.Qt.Key_F5:
            if not self.fullscreenflag:
                self.showFullScreen()
                self.fullscreenflag = True
            else:
                self.showNormal()
                self.fullscreenflag = False

        elif evt.key() == QtCore.Qt.Key_F10:
            currentpage = self.centralWidget().pages.currentWidget()
            if hasattr(currentpage, 'navigation'):
                if self.navigationvisual:
                    currentpage.navigation.setVisible(False)
                    self.navigationvisual = False
                else:
                    currentpage.navigation.setVisible(True)
                    self.navigationvisual = True
        elif evt.key() == QtCore.Qt.Key_F9:
            if self.menuBar().isVisible():
                self.menuBar().hide()
            else:
                self.menuBar().show()
        elif evt.key() == QtCore.Qt.Key_F8:
            if self.statusbar.isVisible():
                self.statusbar.hide()
            else:
                self.statusbar.show()

    def mousePressEvent(self,event):
        # 鼠标点击事件
        if event.button() == QtCore.Qt.LeftButton:
           self.dragPosition = event.globalPos() - self.frameGeometry().topLeft()
           event.accept()

    def mouseMoveEvent(self,event):
        # 鼠标移动事件
        if event.buttons() ==QtCore.Qt.LeftButton:
            self.move(event.globalPos() - self.dragPosition)
            event.accept()


class SplashScreen(QtGui.QSplashScreen):

    def __init__(self, splash_image):
        super(SplashScreen, self).__init__(splash_image)    # 启动程序的图片
        self.setWindowModality(QtCore.Qt.ApplicationModal)

    def fadeTicker(self, keep_t):
        self.setWindowOpacity(0)
```

```
        t = 0
        while t <= 50:
            newOpacity = self.windowOpacity() + 0.02   # 设置淡入
            if newOpacity > 1:
                break
            self.setWindowOpacity(newOpacity)
            self.show()
            t -= 1
            time.sleep(0.04)
        self.show()
        time.sleep(keep_t)
        t = 0
        while t <= 50:
            newOpacity = self.windowOpacity() - 0.02   # 设置淡出
            if newOpacity < 0:
                self.close()
                break
            self.setWindowOpacity(newOpacity)
            self.show()
            t += 1
            time.sleep(0.04)


if __name__ == '__main__':
    import sys
    app = QtGui.QApplication(sys.argv)
    splash = SplashScreen(QtGui.QPixmap(windowsoptions['splashimg']))
    splash.fadeTicker(0)
    app.processEvents()
    main = MainWindow()
    main.show()
    splash.finish(main)
    sys.exit(app.exec_())
```