Разработка простого расширения для QGIS на Python

Обсудить в форуме Комментариев — 5

Эта страница опубликована в основном списке статей сайта
по адресу http://gis-lab.info/qa/qgis-dev-python.html

Описание процесса создания элементарного расширения для QGIS.

QGIS — свободная пользовательская ГИС, обладающая развитым API и продвинутой системой расширений (модулей). Модули создаются для расширения функциональности программы. QGIS написана на объектно-ориентированном языке высокого уровня C++. Так же в QGIS встроены привязки (bindings), которые реализуют практически весь функционал QGIS на языке Python. Графический интерфейс пользователя QGIS базируется на библиотеках Qt4. Разработка для QGIS может вестись на двух языках, Python и C++. Для разработки на языке Python необходимы привязки PyQt, обеспечивающие взаимодействия Python с Qt4. Помимо расширений библиотеки QGIS могут быть также использованы для разработки отдельных приложений, что подробно описано в статье «Создание приложения на базе набора библиотек QGIS на Python».

Данная статья описывает процесс создания каркаса расширения на языке Python. Разработанный каркас можно в дальнейшем использовать как основу для других расширений, выполняющих нужные функции.

Создание расширения состоит из нескольких этапов:

- Идея
- Создание каркаса
- Разработка необходимой функциональности.
- Тестирование
- Публикация

Задача этой статьи - иллюстрация процесса создания простого расширения, которое:

- добавит новую кнопку на панель инструментов
- добавит подменю в меню расширений
- выполнит простое действие.

Это расширение можно будет использовать как каркас для других будущих, более функциональных расширений. Далее мы более подробно разберем процесс создания по шагам.

Содержание

Получение исходного кода

Код расширения, работающий под QGIS 2.0 можно получить в репозитории GitHub:

```
git clone git@github.com:nextgis/testplugin.git
```

Подготовка

Для разработки расширения понадобится

- Любой текстовый редактор
- Python, лучше версии 2.5-2.7.x (3.x не подойдет)
- QGIS версии 2.0 и выше, для проверки работоспособности расширения.

Если вы установили QGIS с помощью установщика osgeo4w или NextGIS QGIS, то у вас уже есть все необходимое для разработки. Ничего дополнительно устанавливать не нужно.

Разработку можно вести прямо в папке, где хранятся расширения или переместить его в нее после разработки, в QGIS эта папка либо QGIS\python\plugins либо </tt>C:\Documents and Settings\username\.qgis2\python\plugins\</tt>.

В папке с плагинами необходимо создать новую, назвав ее так, чтобы бы по названию можно было легко определить, что именно делает расширение. Назовем нашу папку `testplugin`. Это название следует запомнить, так как оно будет фигурировать в различных компонентах кода.

Перед разработкой, нам нужно создать в этой папке несколько новых, пока пустых текстовых файлов:

- __init__.py — начальная точка, создаёт экземпляр основного класса, который передается в QGIS
- testplugin.py — основной код расширения. Содержит всю информацию о всех действиях расширения.
- resources.qrc — xml файл, создаваемый Qt Designer или вручную и содержащий относительные пути к ресурсам расширения, формам, иконкам и т.п. Компилируется в py перед созданием пакета плагина.
- metadata.txt — описание модуля, содержит информацию о расширении, версию, имя разработчика

Помимо этих файлов, если расширение использует формы, могут также присутствовать файлы:

- form.ui — форма созданная с помощью Qt Designer
- form.py — она же, скомпилированная в виде программы на языке Python

Для примера из данной статьи эти файлы не понадобятся.

Итак, пустые файлы созданы и лежат в нужной папке.

Перед разработкой, отредактируем файл metadata.txt. Этот файл содержит информацию о расширении, такую как его название name, т.е. то, как он будет показываться в Менеджере модулей, его описание description, показывается там же. Нужно также указать минимальную версию QGIS, для которой разработано это расширение (при попытке загрузки в QGIS меньшей версии плагин будет отключен) и другие параметры:

```
general
name=TestPlugin
description=This plugin is for testing and templating purposes
category=Vector
version=0.0.2
qgisMinimumVersion=2.0

author=NextGIS
email=info@nextgis.org

icon=icons/icon.png

tags=testing,template

homepage=http://gis-lab.info/qa/qgis-dev-python.html
tracker=https://github.com/nextgis/testplugin/issues
repository=https://github.com/nextgis/testplugin

experimental=True
deprecated=False
```

Разработка

Разработка начинается с файла __init__.py. Главным фрагментом кода является импорт исполняемой части расширения, содержащейся в файле plugin.py (поэтому from testplugin) и содержащей всю содержательную часть кода.

```
def classFactory(iface):
  # Import class TestPlugin from file testplugin.py
  from testplugin import TestPlugin
  return TestPlugin(iface)
```

Название главного импортируемого класса TestPlugin должно быть равно названию класса в коде testplugin.py, название — регистрозависимое, если вызывается TestPlugin, а класс носит название testplugin, будет выдано сообщение об ошибке.

Рассмотрим testplugin.py.

Как и во всех программах на языке Python, все начинается с импорта необходимых для работы классов, в нашем случае это будут классы PyQt — обертки для Qt на языке Python: PyQt4.QtCore и PyQt4.QtGui, активно используемые QGIS и сами классы QGIS: qgis.core

```
# -*- coding: utf-8 -*-
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from qgis.core import *
```

Помимо этого нужно также импортировать ресурсы самого расширения, в нашем случае они будут задаваться через resources.py, как его создать мы разберем чуть позже.

```
import resources
```

После импорта идет раздел главного класса, импортируемого в __init__.py и содержащего все нужные функции. Необходимо обратить внимание, что название класса должно быть равно названию используемому для его вызова в __init__.py. Определение функции начинается с ключевого слова def, за которым следует имя функции и ее аргументы.

Объявим наш основной класс и функцию инициализации:

```
class TestPlugin:

  def __init__(self, iface):
    """Initialize class"""
    # сохраним ссылку на интерфейс QGIS//save reference to QGIS interface
    self.iface = iface
    self.qgsVersion = unicode(QGis.QGIS_VERSION_INT)
```

Одна из важнейших функций — создание элементов интерфейса расширения initGui. При загрузке расширения к пользовательскому интерфейсу iface.self могут добавляться кнопки addToolBarIcon или строки в меню «Модули», addPluginToMenu. Не забываем документировать функции с помощью тройных кавычек.

```
import resources  def initGui(self):
    """Инициализируем графический интерфейс пользователя"""
    #проверим, не пытаются ли запустить плагин из QGIS версии ниже 2.0//check if the
plugin is ran below 2.0
    if int(self.qgsVersion) < 10900:
        qgisVersion = self.qgsVersion[0] + "." + self.qgsVersion[2] + "." +
self.qgsVersion[3]
        QMessageBox.warning(self.iface.mainWindow(),
                            "TestPlugin", "Error",
                            "TestPlugin", "QGIS %s detected.\n" % (qgisVersion) +
                            "TestPlugin", "This version of TestPlugin requires at least
QGIS version 2.0.\nPlugin will not be enabled.")
```

```
        return None

    # create action that will be run by the plugin//создадим действие, которое будет
запускать конфигурацию расширения
    self.action = QAction("Test plugin", self.iface.mainWindow())
    self.action.setIcon(QIcon(":/icons/icon.png"))
    self.action.setWhatsThis("Configuration for test plugin")
    self.action.setStatusTip("This is status tip")

    # добавим пункт в меню инструментов Vector//add plugin menu to Vector toolbar
    self.iface.addPluginToVectorMenu("TestPlugin",self.action)

    # добавим кнопку на панель инструментов Vector//add icon to new menu item in Vector
toolbar
    self.iface.addVectorToolBarIcon(self.action)

    # связь действия с функцией run//connect action to the run method
    self.action.triggered.connect(self.run)
```

Следующая важная функция расширения описывает то, что происходит при его выгрузке, выключении через Менеджер модулей — unload(self)

```
  def unload(self):
    """Действия при выгрузке расширения"""
    # удалить меню расширения и иконку//remove menu and icon from the menu
    self.iface.removeVectorToolBarIcon(self.action)
    self.iface.removePluginVectorMenu("TestPlugin",self.action)
```

И наконец функция выполняющая единственное действие в нашем расширении — run(). Функция создаёт строку сообщения и показывает ее в QMessageBox в главном окне программы.

```
  def run(self):
    """Действия при запуске расширения"""
    # создать и показать сообщение//create a string and show it
    infoString = "This is a test"
    QMessageBox.information(self.iface.mainWindow(),"About",infoString)
```

Какие функции из API доступны через Python

К сожалению, не все функции API QGIS доступны через Python (через C++ доступны все). Чтобы определить, доступны ли конкретные классы и функции, необходимо сначала найти их в документации к API GIS http://doc.qgis.org. А затем в консоли Python (Модули\Консоль Python) определить с помощью команды dir видны ли эти классы и функции из Python.

Например, определим доступные классы:

```
import qgis.core
dir(qgis.core)
```

Результат:

```
'DEFAULT_LINE_WIDTH', 'DEFAULT_POINT_SIZE', 'DEFAULT_SEGMENT_EPSILON',
'ELLPS_PREFIX_LEN', 'GEOCRS_ID', 'GEOPROJ4', 'GEOSRID', 'GEOWKT',
'GEO_EPSG_CRS_AUTHID', 'GEO_EPSG_CRS_ID', 'GEO_NONE', 'LAT_PREFIX_LEN',
'MINIMUM_POINT_SIZE', 'NULL', 'PROJECT_SCALES', 'PROJ_PREFIX_LEN', 'QGis',
'QgsAbstractCacheIndex', 'QgsAbstractFeatureIterator', 'QgsAction',
'QgsAddRemoveItemCommand', 'QgsAddRemoveMultiFrameCommand', 'QgsApplication',
'QgsAtlasComposition', 'QgsAttributeAction', 'QgsAttributeEditorContainer',
'QgsAttributeEditorElement', 'QgsAttributeEditorField', 'QgsBilinearRasterResampler',
'QgsBrightnessContrastFilter', 'QgsBrowserModel', 'QgsCRSCache',
'QgsCacheIndexFeatureId', 'QgsCachedFeatureIterator', 'QgsCachedFeatureWriterIterator',
'QgsCategorizedSymbolRendererV2', 'QgsCentroidFillSymbolLayerV2',
'QgsClipToMinMaxEnhancement', 'QgsClipper', 'QgsColorBrewerPalette',
'QgsColorRampShader', 'QgsComposerArrow', 'QgsComposerAttributeTable',
```

'QgsComposerAttributeTableCompare', 'QgsComposerEffect', 'QgsComposerFrame',
'QgsComposerGroupItem', 'QgsComposerHtml', 'QgsComposerItem', 'QgsComposerItemCommand',
'QgsComposerItemGroup', 'QgsComposerLabel', 'QgsComposerLayerItem',
'QgsComposerLegend', 'QgsComposerLegendItem', 'QgsComposerLegendStyle',
'QgsComposerMap', 'QgsComposerMergeCommand', 'QgsComposerMultiFrame',
'QgsComposerMultiFrameCommand', 'QgsComposerPicture', 'QgsComposerRasterSymbolItem',
'QgsComposerScaleBar', 'QgsComposerShape', 'QgsComposerSymbolV2Item',
'QgsComposerTable', 'QgsComposerTextTable', 'QgsComposition', 'QgsContextHelp',
'QgsContrastEnhancement', 'QgsContrastEnhancementFunction',
'QgsCoordinateReferenceSystem', 'QgsCoordinateTransform',
'QgsCoordinateTransformCache', 'QgsCptCityArchive', 'QgsCptCityBrowserModel',
'QgsCptCityCollectionItem', 'QgsCptCityColorRampItem', 'QgsCptCityColorRampV2',
'QgsCptCityDataItem', 'QgsCptCityDirectoryItem', 'QgsCptCitySelectionItem',
'QgsCredentials', 'QgsCredentialsConsole', 'QgsCsException', 'QgsCubicRasterResampler',
'QgsDataCollectionItem', 'QgsDataDefined', 'QgsDataItem', 'QgsDataProvider',
'QgsDataSourceURI', 'QgsDbFilterProxyModel', 'QgsDiagram',
'QgsDiagramInterpolationSettings', 'QgsDiagramLayerSettings', 'QgsDiagramRendererV2',
'QgsDiagramSettings', 'QgsDirectoryItem', 'QgsDirectoryParamWidget', 'QgsDistanceArea',
'QgsDoubleBoxScaleBarStyle', 'QgsEllipseSymbolLayerV2', 'QgsError', 'QgsErrorItem',
'QgsErrorMessage', 'QgsException', 'QgsExpression', 'QgsFavouritesItem', 'QgsFeature',
'QgsFeatureIterator', 'QgsFeatureRendererV2', 'QgsFeatureRequest', 'QgsFeatureStore',
'QgsField', 'QgsFields', 'QgsFillSymbolLayerV2', 'QgsFillSymbolV2',
'QgsFontMarkerSymbolLayerV2', 'QgsFontUtils', 'QgsGPSConnection',
'QgsGPSConnectionRegistry', 'QgsGPSDetector', 'QgsGPSInformation', 'QgsGeometry',
'QgsGeometryCache', 'QgsGeometryValidator', 'QgsGml', 'QgsGmlFeatureClass',
'QgsGmlSchema', 'QgsGpsdConnection', 'QgsGradientStop', 'QgsGraduatedSymbolRendererV2',
'QgsHistogramDiagram', 'QgsHttpTransaction', 'QgsHueSaturationFilter',
'QgsImageFillSymbolLayer', 'QgsLabel', 'QgsLabelAttributes', 'QgsLabelCandidate',
'QgsLabelComponent', 'QgsLabelPosition', 'QgsLabelSearchTree',
'QgsLabelingEngineInterface', 'QgsLayerItem', 'QgsLegendModel',
'QgsLinePatternFillSymbolLayer', 'QgsLineSymbolLayerV2', 'QgsLineSymbolV2',
'QgsLinearMinMaxEnhancement', 'QgsLinearMinMaxEnhancementWithClip',
'QgsLinearlyInterpolatedDiagramRenderer', 'QgsLogger', 'QgsMapLayer',
'QgsMapLayerRegistry', 'QgsMapRenderer', 'QgsMapToPixel', 'QgsMarkerLineSymbolLayerV2',
'QgsMarkerSymbolLayerV2', 'QgsMarkerSymbolV2', 'QgsMessageLog', 'QgsMessageLogConsole',
'QgsMessageOutput', 'QgsMessageOutputConsole', 'QgsMimeDataUtils',
'QgsMultiBandColorRenderer', 'QgsNMEAConnection', 'QgsNetworkAccessManager',
'QgsNumericScaleBarStyle', 'QgsOWSConnection', 'QgsOfflineEditing', 'QgsOgcUtils',
'QgsPaintEngineHack', 'QgsPalLabeling', 'QgsPalLayerSettings',
'QgsPalettedRasterRenderer', 'QgsPaperItem', 'QgsPieDiagram', 'QgsPluginLayer',
'QgsPluginLayerRegistry', 'QgsPluginLayerType', 'QgsPoint',
'QgsPointDisplacementRenderer', 'QgsPointPatternFillSymbolLayer', 'QgsProject',
'QgsProjectBadLayerDefaultHandler', 'QgsProjectBadLayerHandler',
'QgsProjectFileTransform', 'QgsProjectVersion', 'QgsProperty', 'QgsPropertyKey',
'QgsPropertyValue', 'QgsProviderCountCalcEvent', 'QgsProviderExtentCalcEvent',
'QgsProviderMetadata', 'QgsProviderRegistry', 'QgsPseudoColorShader',
'QgsPythonRunner', 'QgsRaster', 'QgsRasterBandStats', 'QgsRasterBlock',
'QgsRasterChecker', 'QgsRasterDataProvider', 'QgsRasterDrawer', 'QgsRasterFileWriter',
'QgsRasterHistogram', 'QgsRasterIdentifyResult', 'QgsRasterInterface',
'QgsRasterIterator', 'QgsRasterLayer', 'QgsRasterNuller', 'QgsRasterPipe',
'QgsRasterProjector', 'QgsRasterPyramid', 'QgsRasterRange', 'QgsRasterRenderer',
'QgsRasterResampleFilter', 'QgsRasterResampler', 'QgsRasterShader',
'QgsRasterShaderFunction', 'QgsRasterTransparency', 'QgsRasterViewPort',
'QgsRectangle', 'QgsRenderChecker', 'QgsRenderContext', 'QgsRendererCategoryV2',
'QgsRendererRangeV2', 'QgsRendererV2AbstractMetadata', 'QgsRendererV2Metadata',
'QgsRendererV2Registry', 'QgsRuleBasedRendererV2', 'QgsRunProcess',
'QgsSVGFillSymbolLayer', 'QgsSatelliteInfo', 'QgsScaleBarStyle', 'QgsScaleCalculator',
'QgsScaleUtils', 'QgsSimpleFillSymbolLayerV2', 'QgsSimpleLineSymbolLayerV2',
'QgsSimpleMarkerSymbolLayerV2', 'QgsSingleBandColorDataRenderer',
'QgsSingleBandGrayRenderer', 'QgsSingleBandPseudoColorRenderer',
'QgsSingleBoxScaleBarStyle', 'QgsSingleCategoryDiagramRenderer',
'QgsSingleSymbolRendererV2', 'QgsSnapper', 'QgsSnappingResult', 'QgsSpatialIndex',
'QgsStyleV2', 'QgsSvgCache', 'QgsSvgCacheEntry', 'QgsSvgMarkerSymbolLayerV2',
'QgsSymbolLayerV2', 'QgsSymbolLayerV2AbstractMetadata', 'QgsSymbolLayerV2Metadata',
'QgsSymbolLayerV2Registry', 'QgsSymbolLayerV2Utils', 'QgsSymbolV2',
'QgsSymbolV2LevelItem', 'QgsSymbolV2RenderContext', 'QgsSymbologyV2Conversion',
'QgsTextDiagram', 'QgsTicksScaleBarStyle', 'QgsTolerance',
'QgsVectorColorBrewerColorRampV2', 'QgsVectorColorRampV2', 'QgsVectorDataProvider',

```
'QgsVectorFieldSymbolLayer', 'QgsVectorFileWriter', 'QgsVectorGradientColorRampV2',
'QgsVectorJoinInfo', 'QgsVectorLayer', 'QgsVectorLayerCache',
'QgsVectorLayerEditBuffer', 'QgsVectorLayerEditUtils', 'QgsVectorLayerFeatureIterator',
'QgsVectorLayerImport', 'QgsVectorLayerJoinBuffer', 'QgsVectorRandomColorRampV2',
'QgsZipItem', 'USER_CRS_START_ID', '__doc__', '__file__', '__name__', '__package__'
```

Далее, после выбора нужного класса, например QgsFeature, посмотрим его методы:

```
dir(qgis.core.QgsFeature)
```

Результат:

```
'__class__', '__delattr__', '__delitem__', '__dict__', '__doc__', '__format__',
'__getattribute__', '__getitem__', '__hash__', '__init__', '__iter__', '__module__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'attribute', 'attributes',
'deleteAttribute', 'fieldNameIndex', 'fields', 'geometry', 'geometryAndOwnership',
'id', 'initAttributes', 'isValid', 'setAttribute', 'setAttributes', 'setFeatureId',
'setFields', 'setGeometry', 'setGeometryAndOwnership', 'setValid'
```

Ресурсы

Для разработки расширения понадобится создать специальный файл, который будет содержать указатели на используемые расширением ресурсы, такие как например иконки. Пример такого файла:

```
<RCC>
<qresource prefix="" >
    <file>icons/icon.png</file>
</qresource>
</RCC>
```

Рекомендуется использовать уникальный qresource prefix, который не будет конфликтовать с другими расширениями.

После создания этого файла, необходимо его скомпилировать с помощью pyrcc4 в формат, который можно импортировать с помощью Python:

```
pyrcc4 -o resources.py resources.qrc
```

И как уже упоминалось выше, импортировать получившиеся ресурсы в основном коде testplugin.py.

Заключение

Разработка каркаса нашего расширения завершена. Если она велась в папке расширений, то после запуска QGIS расширение должно автоматически появиться в списке расширений и можно начать его тестирование, отладку и, конечно, наполнение нужными полезными функциями.

Помните, что лучшим пособием по разработке для QGIS являются расширения созданные другими авторами.

Ссылки по теме

- Создание приложения на базе набора библиотек QGIS на Python
- Организация и работа с репозиториями расширений QGIS
- Установка модулей расширения в QGIS
- Документация по QGIS

Обсудить в форуме Комментариев — 5

Последнее обновление: 2014-05-15 00:51

Дата создания: 19.02.2009
Автор(ы): Максим Дубинин