Добавление списка слоёв в приложение PyQGIS

Обсудить в форуме Комментариев — 38

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

Подробное описание процесса подключения списка слоёв в приложениях на базе QGIS с помощью Python.

QGIS помимо готовой пользовательской ГИС является еще и набором библиотек, которые могут быть использованы для создания новых приложений. Мы уже рассказывали о создании самостоятельного отдельного приложения на основе QGIS. К сожалению, при разработке таких приложений некоторые элементы интерфейса необходимо создавать самостоятельно. Один из примеров такого элемента — список загруженных слоёв (иногда неверно называемый легендой).

Данная статья описывает процесс встраивания списка слоёв в приложение PyQGIS. В основе статьи лежит перевод поста «Layer list widget for PyQGIS applications», сделанного German Carrillo в блоге GeoTux.

Работоспособность примеров проверена на QGIS 1.7. В новых версиях QGIS могут быть внесены изменения, делающие данные примеры неработоспособным.

Содержание

- 1 Подготовка
- 2 Общие сведения о виджете
  - 2.1 Возможности
- 3 Подключение легенды к приложению
  - 3.1 Инициализация
  - 3.2 Использование
- 4 Ссылки по теме

Подготовка

Будем исходить из того, что все программное обеспечение установлено при помощи установщика OSGeo4W (подробнее) используя путь по умолчанию C:\OSGeo4W. Кроме самой QGIS со всеми зависимостями, необходимо выбрать следующие пакеты в категории Libs:

```
pyqt4, qt4-devel, qt4-doc, qt4-libs
```

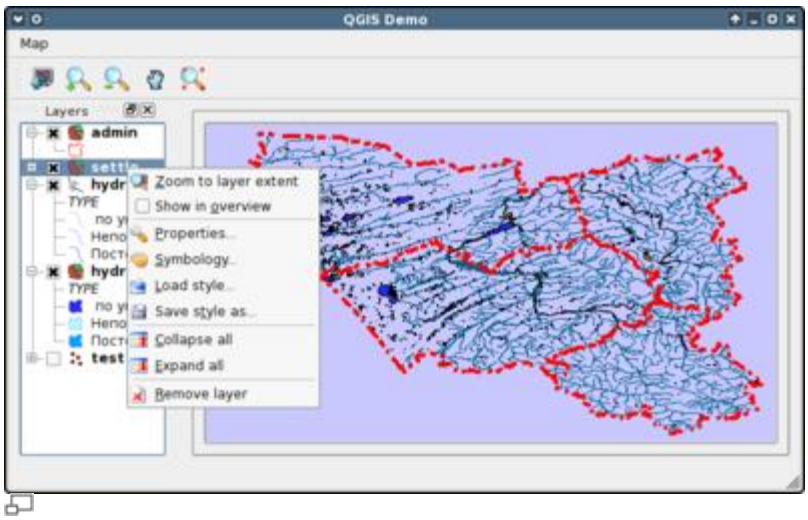
В этих пакетах находятся инструменты разработчика, необходимые библиотеки и документация.

Предполагается, что читатель уже знаком с процессом создания приложений PyQGIS и QGIS Python API. Если это не так, рекомендуем ознакомиться со статьей «Создание приложения на базе набора библиотек QGIS на Python». Для демонстрации основных аспектов работы со списком слоёв мы немного модифицировали пример, рассмотренный в этой статье. Скачать полный код приложения можно здесь.

Общие сведения о виджете

Рассматриваемый виджет разработан German Carrillo и является упрощенной версией списка слоёв QGIS. Основное отличие от оригинала — не поддерживается новая символика и отсутствуют группы слоёв.

Вот так выглядит приложение со списком слоёв:



Возможности

- получение упорядоченного списка слоёв
- изменение порядка следования слоёв в легенде с соответствующим изменением порядка слоёв карты
- управление видимостью слоя
- отображение/изменение символики слоя (рекомендуется создать свой собственный диалог настройки символики, автор виджета предлагает лишь очень простой пример)
- изменение некоторых параметров слоя
- загрузка/сохранение стилей (.qml)
- удаление слоёв
- интеграция с картой и инструментами, действующими на загруженные слои (Скрыть/Показать все, Удалить все)

Виджет распространяется по условиям GNU GPL v2.

Подключение легенды к приложению

Сначала нужно загрузить архив и извлечь его содержимое в каталог с приложением. В архиве находится:

- файл legend.py, который реализует весь фукционал легенды
- несколько иконок, необходимых для отображения слоёв разных типов
- иконки действий для выпадающего меню
- диалог настройки свойств слоя

Инициализация

В основном классе нашего приложения импортируем класс Legend из файла legend.py

```
from legend import Legend
```

В этом же классе создадим новый метод CreateLegendWidget, который будет отвечать за создание плавающего окна и за установку связи между легендой и картой.

```
def createLegendWidget( self ):
  # создаем легенду
  self.legend = Legend( self )
  self.legend.setCanvas( self.canvas )
  self.legend.setObjectName( "theMapLegend" )

  # создаем плавающую панель и добавляем к ней легенду
  self.LegendDock = QDockWidget( "Layers", self )
  self.LegendDock.setObjectName( "legend" )
  self.LegendDock.setAllowedAreas( Qt.LeftDockWidgetArea | Qt.RightDockWidgetArea )
  self.LegendDock.setWidget( self.legend )
  self.LegendDock.setContentsMargins ( 9, 9, 9, 9 )
  self.addDockWidget( Qt.LeftDockWidgetArea, self.LegendDock )
```

Вызов этого метода необходимо добавить в метод __init__ основного класса приложения. Обратите внимание, что создание легенды необходимо выполнять после создания карты (QgsMapCanvas).

```
class MainWindow( QMainWindow, Ui_MainWindow ):
  def __init__( self ):
    MainWindow.__init__( self )

    # Требуется Qt4 для инициализации пользовательского интерфейса
    self.setupUi( self )
    # Создаем карту
    self.canvas = QgsMapCanvas()
    ...
    # создаем легенду
    self.createLegendWidget()
```

Использование

Виджет легенды использует несколько иконок: иконки для разных типов слоёв и иконки для контекстного меню слоя. Чтобы они отображались необходимо скомпилировать входящий в архив файл resources.qrc командой

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

В файле legend.py уже присутствует ссылка на сгенерированный таким образом файл. Если приложение использует свой собственный ресурсный файл (а так чаще всего и бывает), есть два варианта

1. переименовать один из ресурсных файлов
2. поместить все ресурсы в один файл

В последнем случае может потребоваться задание раздельных префиксов для ресурсов, относящихся к разным частям приложения (например, один префикс для иконок легенды и второй для остальных файлов). При изменении префикса для ресурсов легенды также необходимо соответствующим образом изменить переменную resource_prefix в файле legend.py

```
resources_prefix = ":/legend/imgs/"
```

Мы выбрали второй вариант: все изображения находятся в каталоге images и вложенных каталогах, в ресурсном файле создано два префикса и соотвественно изменена переменная resource_prefix в файле legend.py. Обратите внимание, что помимо задания другого префикса, для каждого файла был создан псевдоним (alias).

При использовании легенды немного меняется процедура добавления слоёв. Если раньше для загрузки и отображения слоя мы использовали свой собственный метод, в котором добавляли слой и обновляли LayerSet, то теперь контролировать процесс загрузки и удаления будет виджет легенды. Виджет имеет слот, обновляющий список слоёв при каждом добавлении, удалении или перемещении слоя. Всё что от нас требуется — вызвать соответствующие методы класса QgsMapLayerRegistry (например, addMapLayer или removeMapLayer). А вот вызов метода setLayerSet класса QgsMapCanvas теперь не нужен, т.к. об этом заботится виджет легенды. Его можно закомментировать или вообще удалить. Таким образом, для добавления нового слоя необходима всего одна строка кода

```
QgsMapLayerRegistry.instance().addMapLayer( layer )
```

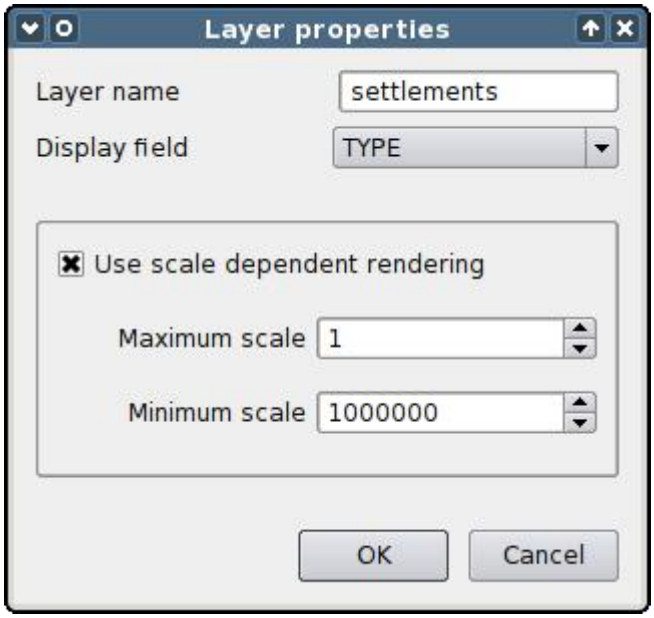
здесь layer — экземпляр класса QgsVectorLayer или QgsRasterLayer, подробнее об этом можно прочесть в предыдущей статье.

Вместе в легендой поставляется простой диалог свойств слоя. Чтобы им можно было пользоваться, необходимо скомпилировать файл dlgLayerProperties.ui

```
pyuic4 dlgLayerProperties.ui -o dlgLayerProperties_ui.py
```

Диалог позволяет задать имя слоя, изменить отображаемое поле и настраивать видимость в пределах масштаба. Если этого функционала не достаточно, можно, взяв за основу существующий диалог, разработать

свой вариант.



При смене активного слоя виджет легенды испускает сигнал activeLayerChanged, который можно использовать для активации/деактивации определенных инструментов приложения, например, в зависимости от типа слоя (растр или вектор). Для этого необходимо в главном классе приложения связать сигнал со слотом, который и будет отвечать за активацию/деактивацию инструментов или выполнять какие-то другие действия. Например, так

```
self.connect( self.legend, SIGNAL( "activeLayerChanged" ), self.enableTools )

def enableTools( self ):
  if not self.legend.activeLayer():
    # нет активного слоя, деактивируем инструменты общего назначения
  else:
    # делаем инструменты общего назначения доступными

    layerType = self.legend.activeLayer().layer().type()
    if layerType == 0: # Vector Layer
      # активируем инструменты для работы с вектором и деактивируем остальные
    elif layerType == 1: # Raster Layer
      # активируем инструменты для работы с растром и деактивируем остальные
```

Также виджет предоставляет несколько полезных методов, с помощью которых можно

- перебрать все слои

```
for layer in self.legend.layers:
  # сделать что-то со слоем
```

- скрыть или отобразить все слои

```
def showAllLayers( self ):
  self.legend.setStatusForAllLayers( True )
```

- получить активный слой

```
return self.legend.activeLayer()
```

- обновить символику слоя

```
self.legend.refreshLayerSymbology( layer )
```

- удалить слои по идентификаторам

```
self.legend.removeLayers( layerIds )
```

Разумеется, это далеко не полный список методов рассматриваемого виджета. Более полное представление о его возможностях можно получить заглянув в файл legend.py, код достаточно прозрачный и снабжен комментариями, так что разобраться в нем не составит труда.

На этом знакомство с виджетом легенды будем считать оконченным. Многое осталось за бортом: например, не были затронуты вопросы настройки символики, расширения диалогового окна свойств слоя,

взаимодействие легенды и инструментов карты... Возможно, если тема окажется интересной, статья получит продолжение.

Исходные коды примера, а так же все необходимые изображения и тестовый набор данных можно скачать одним файлом.

Необходимо отметить, что существует еще один вариант легенды, созданный Aaron Racicot для проекта OpenOceanMap и основанный на элементе управления QCheckBox, а не на QTreeWidget.

Ссылки по теме

- Создание приложения на базе набора библиотек QGIS на Python
- Layer list widget for PyQGIS applications
- Документация по QGIS

Обсудить в форуме Комментариев — 38

Последнее обновление: 2014-05-15 00:14

Дата создания: 14.02.2011
Автор(ы): Александр Бруй