

基于 QGIS 的 GIS 原型系统

王一伊

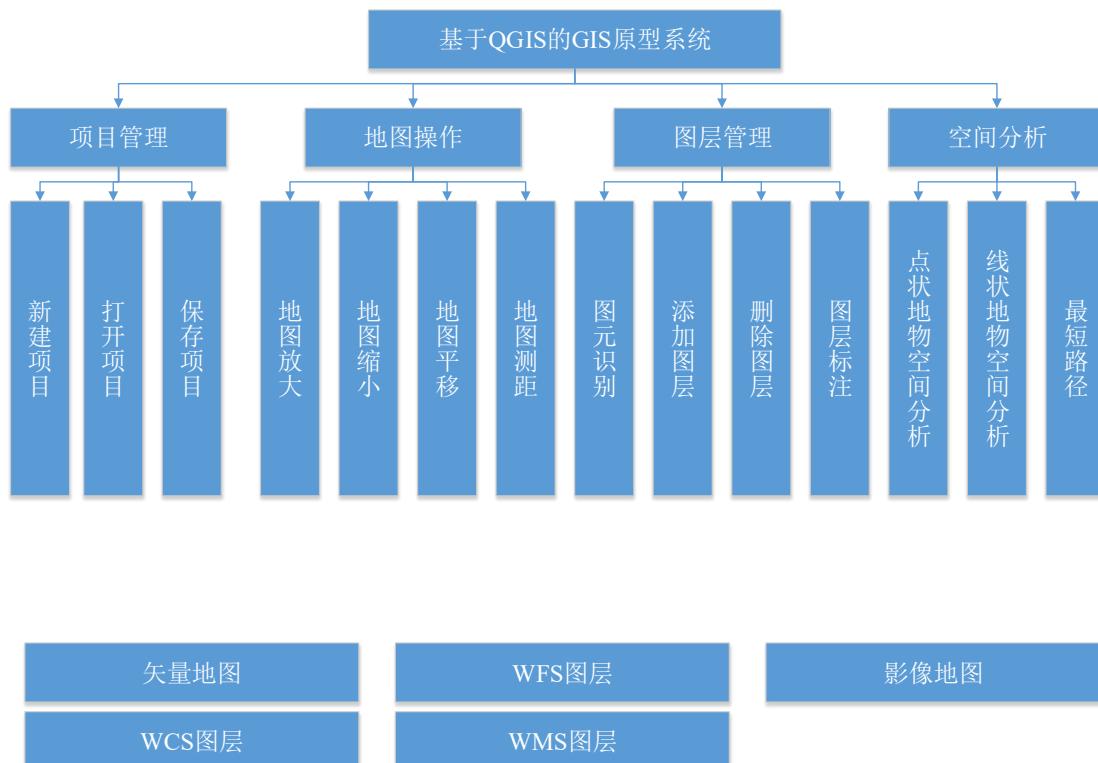
1.实验要求

1.1 基本要求

主要目标是利用开源软件库 QGIS 实现基本 GIS 系统。当前 QGIS 以两种方式供用户使用：对于终端用户，以应用程序的方式供用户使用；对于编程者来说，以动态库等方式供用户进行二次开发。本次设计主要是以第二种方式进行，并且不可在 QGIS 提供的 QGIS 应用程序的源代码上修改，必须基于 QGIS 库新建工程来实现。

1.2 功能要求

主要完成**基于 QGIS 的 GIS 原型系统**，系统功能主要包括项目管理、地图操作、图层管理和空间分析模块。系统具体功能如下图所示。



项目管理、地图操作、图层管理功能模块的相关功能描述详见 QGIS 帮助文档。对于图层管理模块需要实现矢量地图、影像地图、WCS 图层、WFS 图层、

WMS 图层这 5 种地图图层的管理。

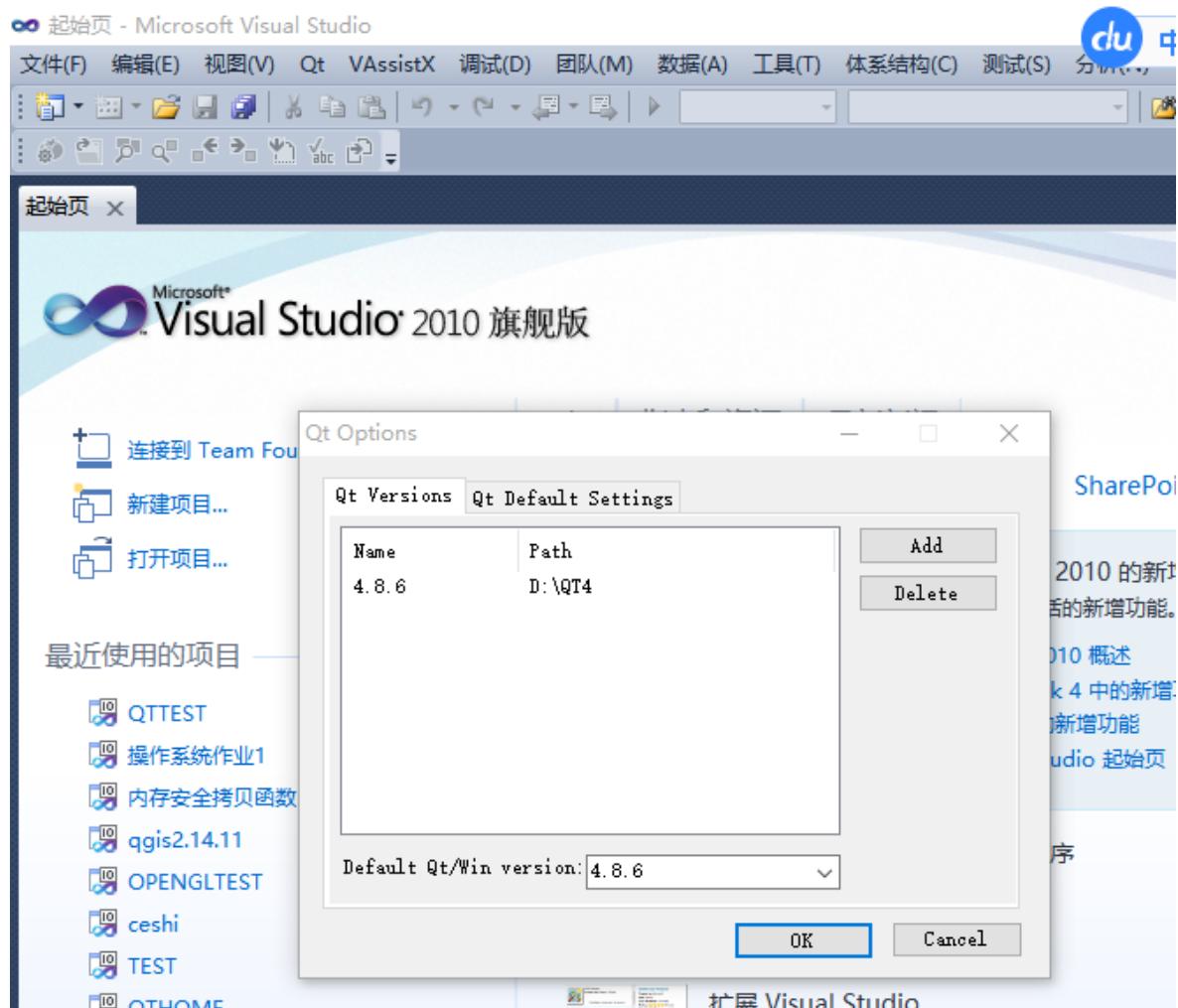
2. QGIS 二次开发步骤的说明（QGIS 编译请见编译文档）

2.1 新建工程

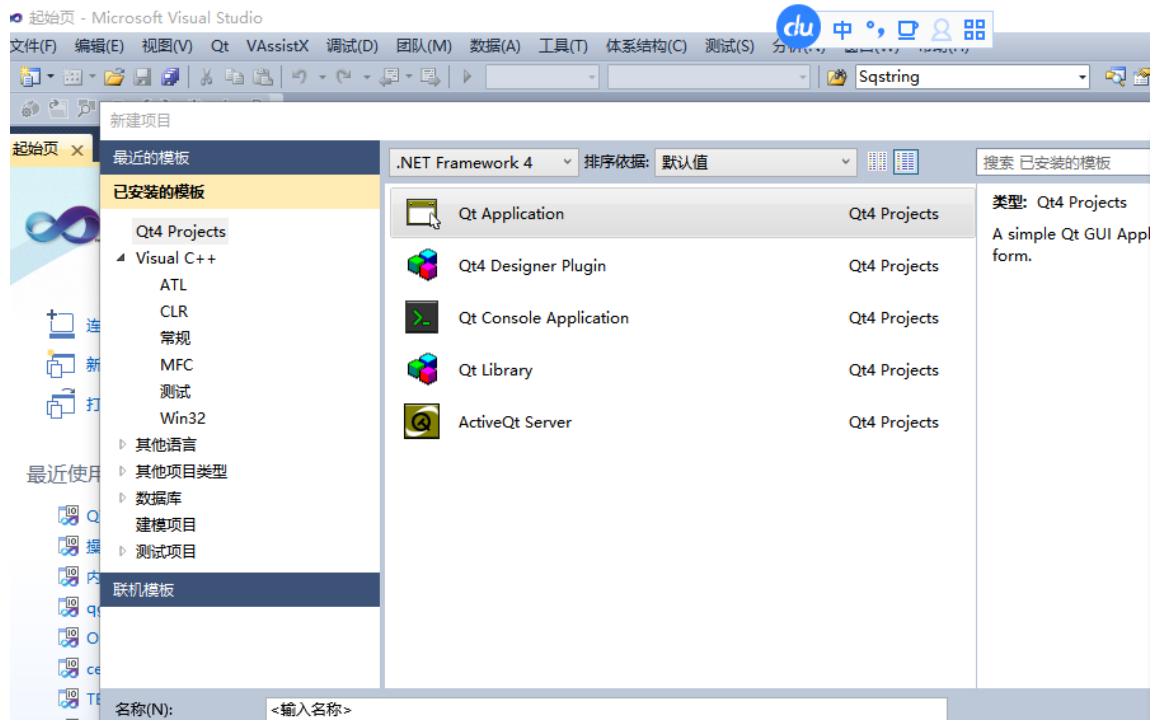
版本说明：QT4.8.6，VS2010

在完成 qgis2.14.11 源码编译后，可以开始编写 GIS 工程了。

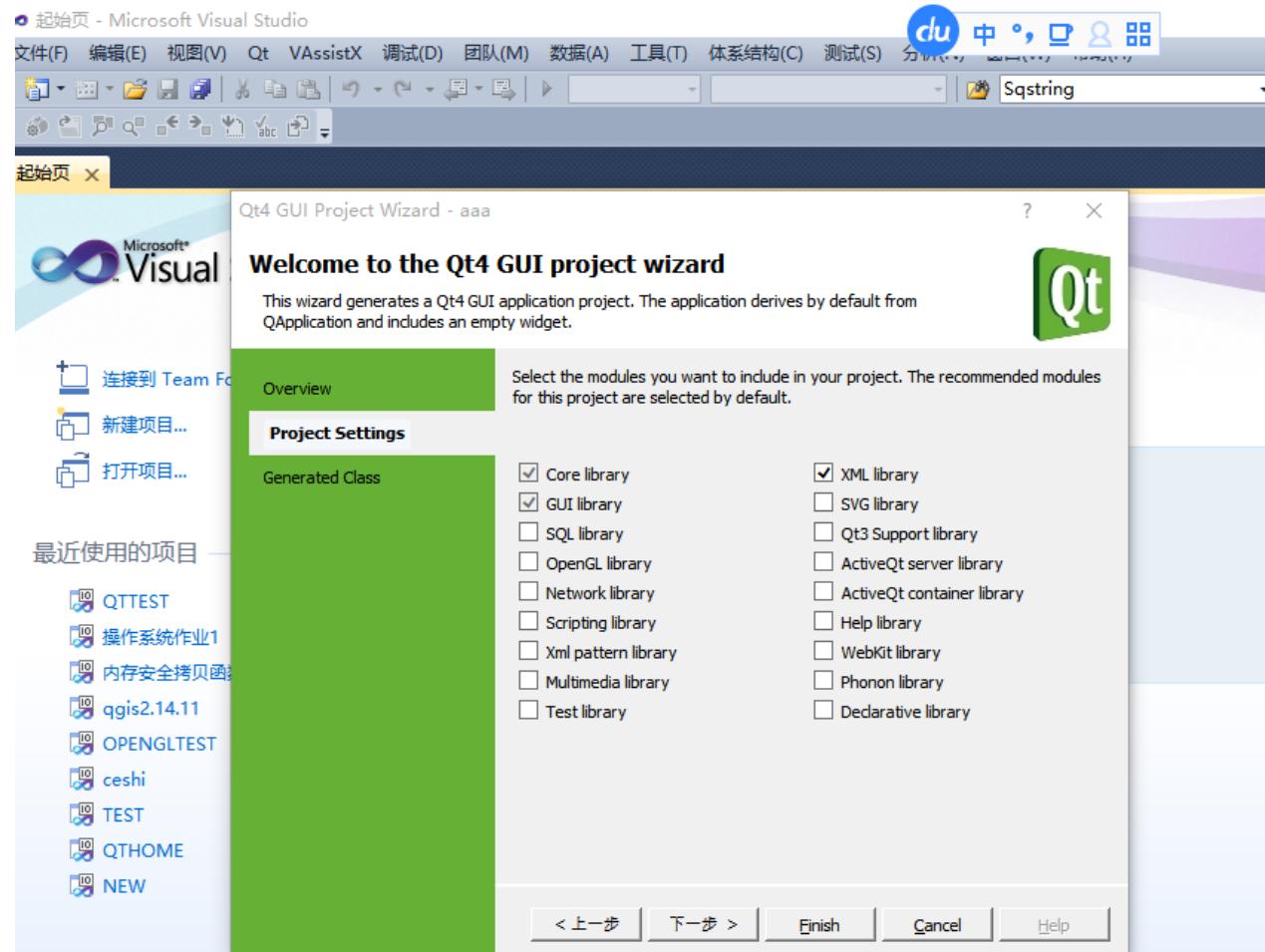
首先在 VS2010 中配置 QT4 需要一个插件 qt-vs2010-addin，于是我百度下载了一个，将 QT4 添加到 VS2010 中。



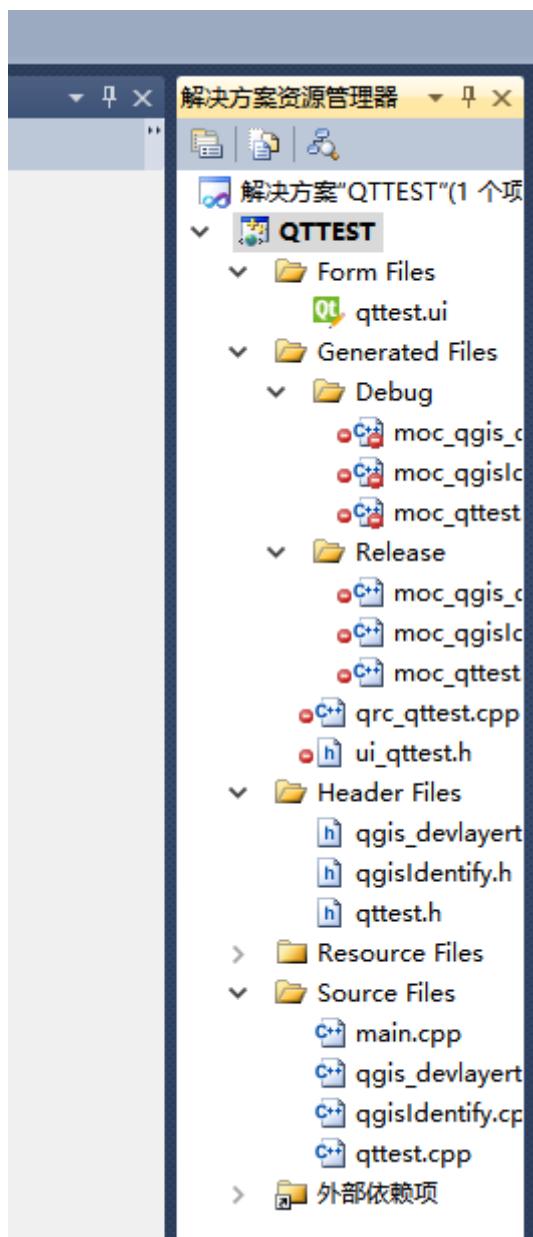
点击新建工程，选择 QT4 projects, 创建一个 QT Application



在新建选项中，注意一定要勾选 XML library，否则会报错。因为 QgsMapCanvas 需要用到 QDomNode 文件，而 QDomNode 是一个 XML 文件，包含在 Qt 的 XML library 中。



新建项目后会得到一个 QT 的工程

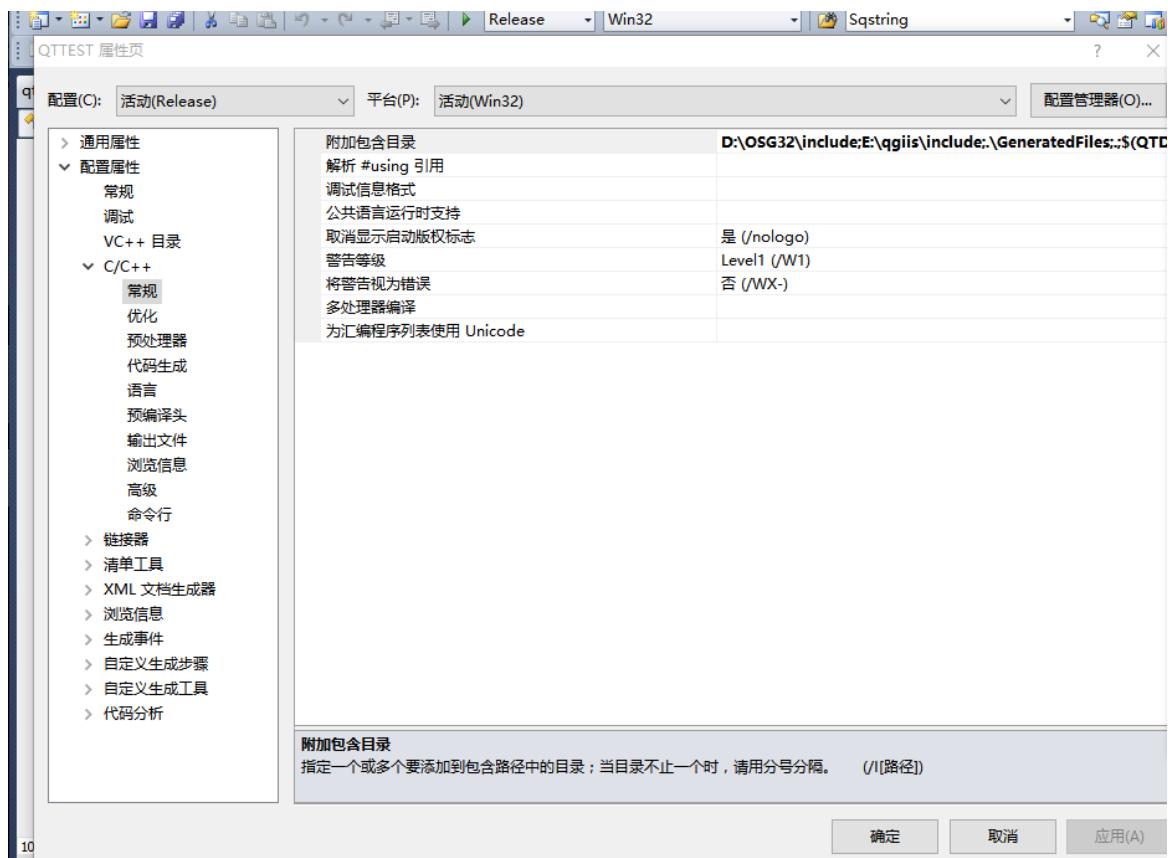


2.2 工程配置

由于需要利用编译好的 qgis 进行二次开发，因此进行工程配置。（要选择 Release 模式）在工程项目上右键属性打开属性配置界面。【以下配置具体在哪个选项卡中请直接看图。】

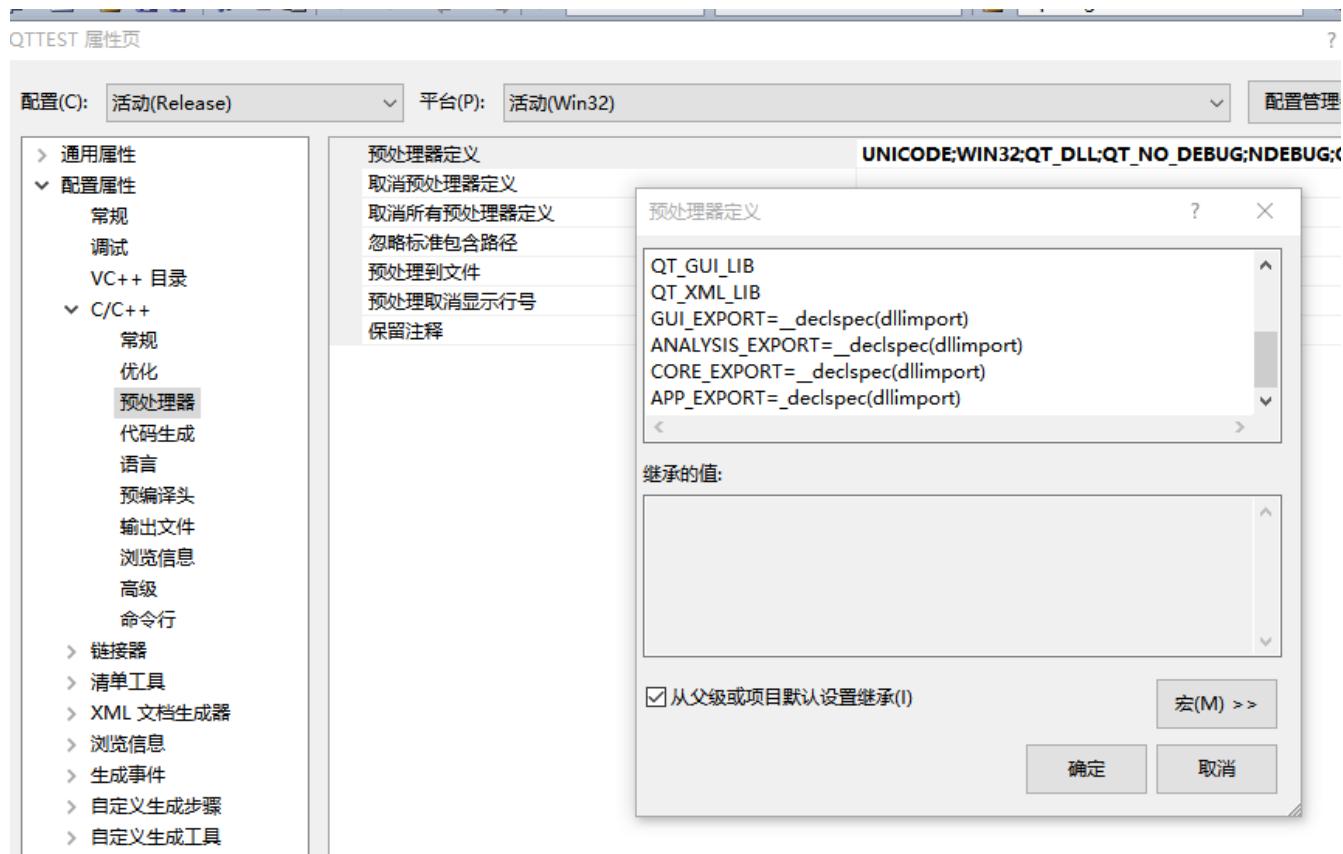
2.2.1 配置附加包含目录

这里需要添加 OSGeo4W 的 include，以及通过编译 qgis 的 INSTALL 文件新生成的 qgis 文件的 include。



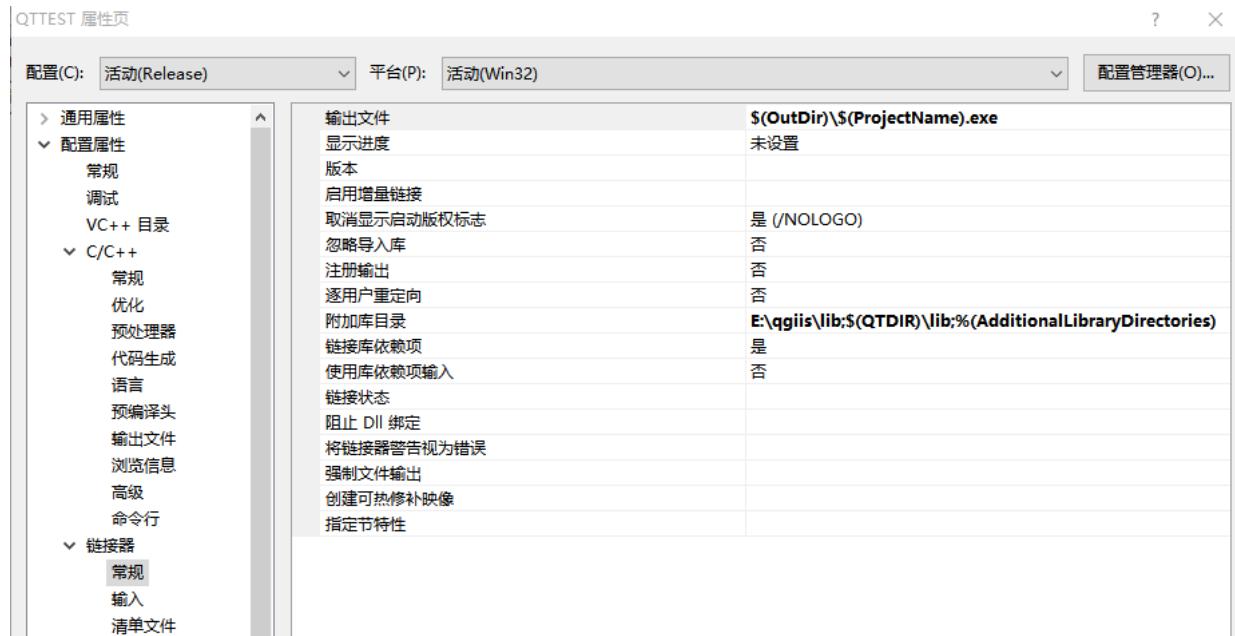
2.2.2 添加预处理命令

添加后四行预处理，因为 release 版本无预处理定义，不加预处理会报错。



2.2.3 添加附加库目录

路径为 INSTALL 生成的 qgis 的 lib。



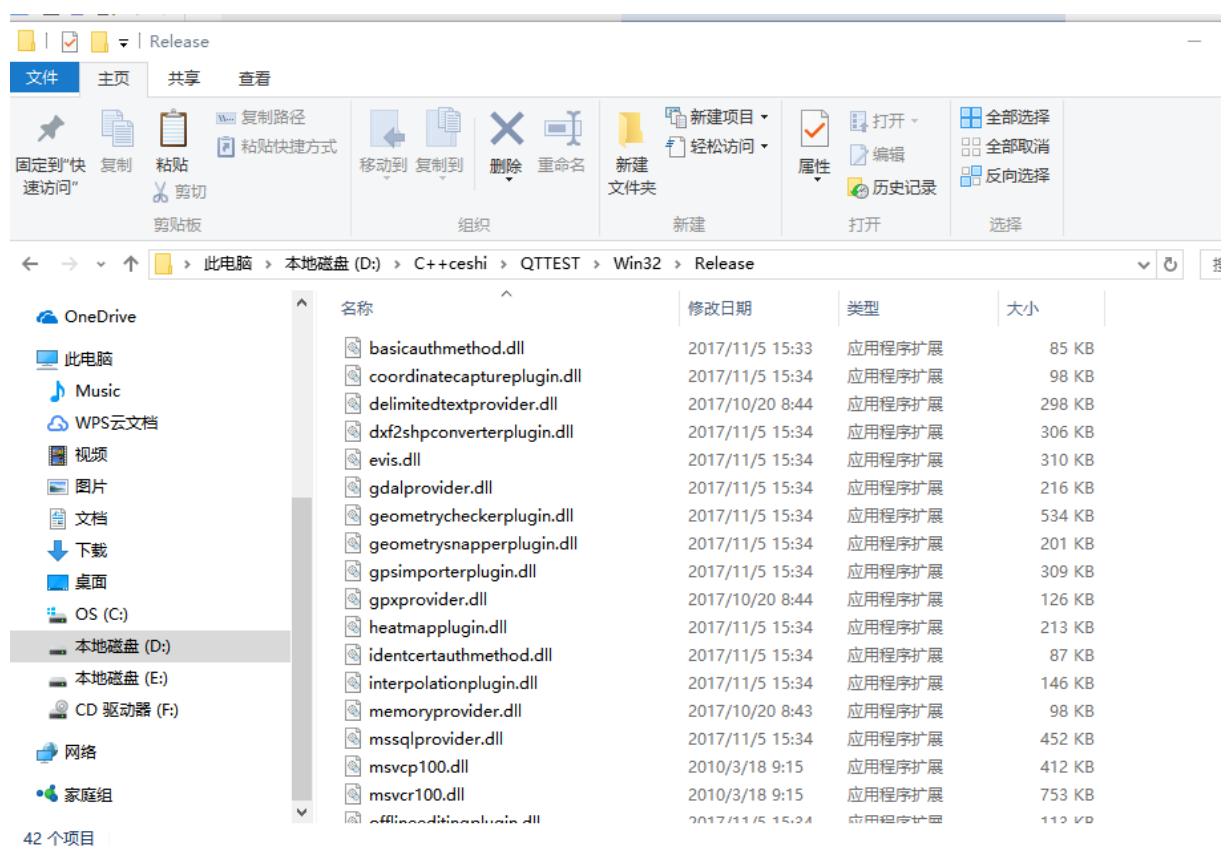
2.2.4 添加附加依赖项

Qgis_analysis.lib, Qgis_core.lib, Qgis_gui.lib。

QTTEST 属性页



2.2.5 将 qgis 相关的 dll 拷贝一份放到新建的工程目录下，才能正确的运行。



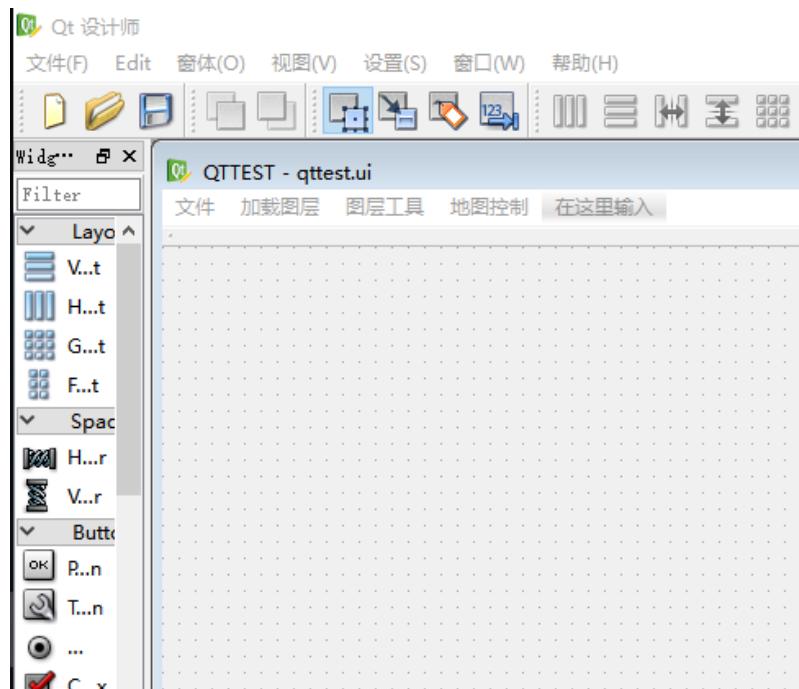
 qgis_analysis.dll	2017/10/19 16:59	应用程序扩展
 qgis_app.dll	2017/11/5 15:33	应用程序扩展
 qgis_core.dll	2017/10/19 16:55	应用程序扩展
 qgis_gui.dll	2017/11/5 15:22	应用程序扩展
 qgis_networkanalysis.dll	2017/10/20 8:42	应用程序扩展
 QTTEST.exe	2017/12/31 13:44	应用程序
 rasterterrainplugin.dll	2017/11/5 15:33	应用程序扩展
 roadgraphplugin.dll	2017/11/5 15:33	应用程序扩展
 spatialiteprovider.dll	2017/11/5 15:33	应用程序扩展
 spatialqueryplugin.dll	2017/11/5 15:33	应用程序扩展
 topolplugin.dll	2017/11/5 15:33	应用程序扩展
 virtuallayerprovider.dll	2017/11/5 15:33	应用程序扩展
 wcsprovider.dll	2017/11/5 15:33	应用程序扩展
 wfsprovider.dll	2017/11/5 15:33	应用程序扩展
 wmsprovider.dll	2017/11/5 15:33	应用程序扩展
 zonalstatisticsplugin.dll	2017/11/5 15:33	应用程序扩展
<hr/>		

配置好以上内容之后就可以写代码进行测试了。我在配置好工程后的第一个测试方式是看编写好的加载图层代码能否正确的加载图层。

3. 基于 QGIS 的 GIS 原型系统的开发说明

题外话，如何找到功能对应的实现类？我是通过用 VS 调试执行 qgis.exe 然后点击相应功能按键，再看调试窗口显示它调用了哪些类，进而可以去读对应的 QGIS 的 API 文档，找到实现功能的函数，最后进行代码实现。

首先，界面设计可以通过 QT designer 实现：

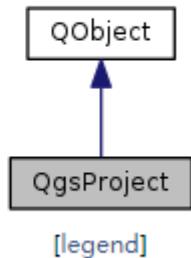


每添加一个子菜单按钮都会自动生成一个对应的动作对象，用于响应点击动作。

<input type="checkbox"/> actionRaster	<input checked="" type="checkbox"/>	Add_Raster
<input type="checkbox"/> actionVector	<input checked="" type="checkbox"/>	Add_Vector
<input type="checkbox"/> action	<input checked="" type="checkbox"/>	新建项目
<input type="checkbox"/> action_2	<input checked="" type="checkbox"/>	打开项目
<input type="checkbox"/> action_4	<input checked="" type="checkbox"/>	保存项目
<input type="checkbox"/> action_GIS	<input checked="" type="checkbox"/>	退出GIS
<input type="checkbox"/> action_WMS	<input checked="" type="checkbox"/>	Add_WMS
<input type="checkbox"/> action_WCS	<input checked="" type="checkbox"/>	Add_WCS
<input type="checkbox"/> action_WFS	<input checked="" type="checkbox"/>	Add_WFS
<input type="checkbox"/> actionAttribute	<input checked="" type="checkbox"/>	identify feature
<input type="checkbox"/> actionDelete	<input checked="" type="checkbox"/>	delete
<input type="checkbox"/> actionLabel	<input checked="" type="checkbox"/>	label
<input type="checkbox"/> actionCalculate	<input checked="" type="checkbox"/>	caculate
<input type="checkbox"/> actionTool	<input checked="" type="checkbox"/>	dissmissTool
<input type="checkbox"/> actionL	<input checked="" type="checkbox"/>	bigger
<input type="checkbox"/> actionAller	<input checked="" type="checkbox"/>	smaller

3.1 项目管理

在项目管理的实现上我阅读了 qgis 的官方 API 中的 QgsProject 类。



该类中包含了 read(), write() 函数，能够自动的读写. qgs 项目工程文件。

```
bool read (const QFileInfo &file)
    Read project file. More...


---


bool read ()
    presuming that the caller has already reset the


---


bool write (const QFileInfo &file)
    Write project file. More...


---


bool write ()
```

clear() 函数能够清空 qgs 项目，也就是实现了新建项目的功能。

```
void clear ()
    Clear the project. More...
```

因此利用以上三个函数能够实现 qgs 项目的新建、打开、保存。

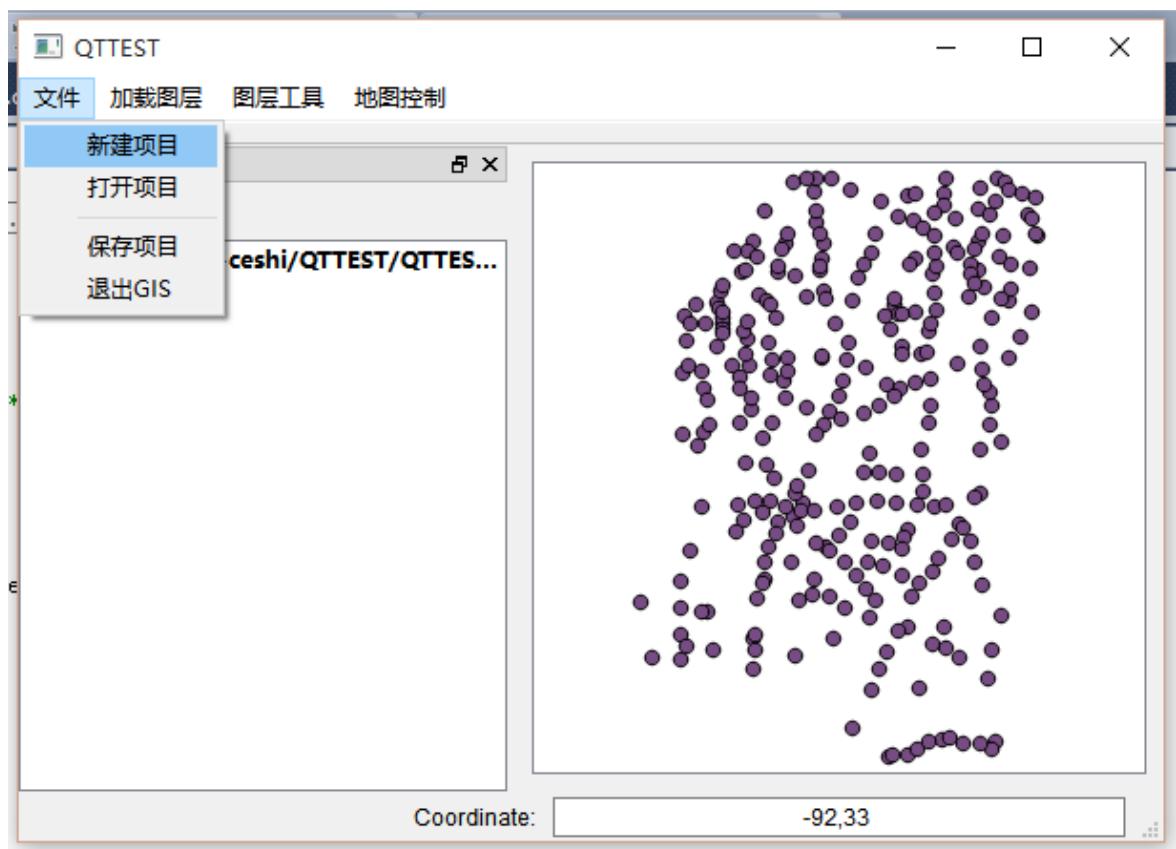
3.1.1 新建项目

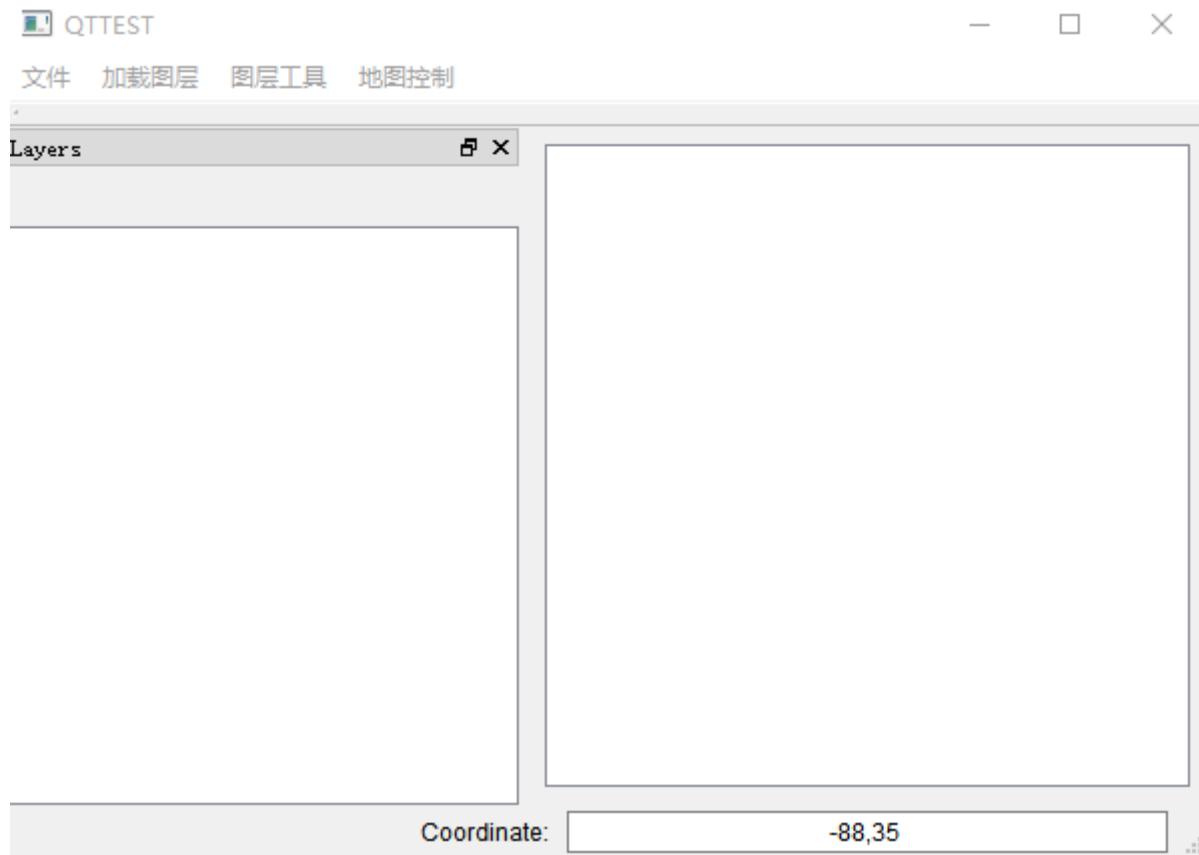
实现代码：

调用 Qgsproject 的 clear() 方法能够清空项目，新建一个空白项目。

```
//*****
//项目管理
//新建项目
void QTTEST::newfile()
{
    QgsProject::instance()->clear(); //清空项目
}
```

实现效果：





3.1.2 打开项目

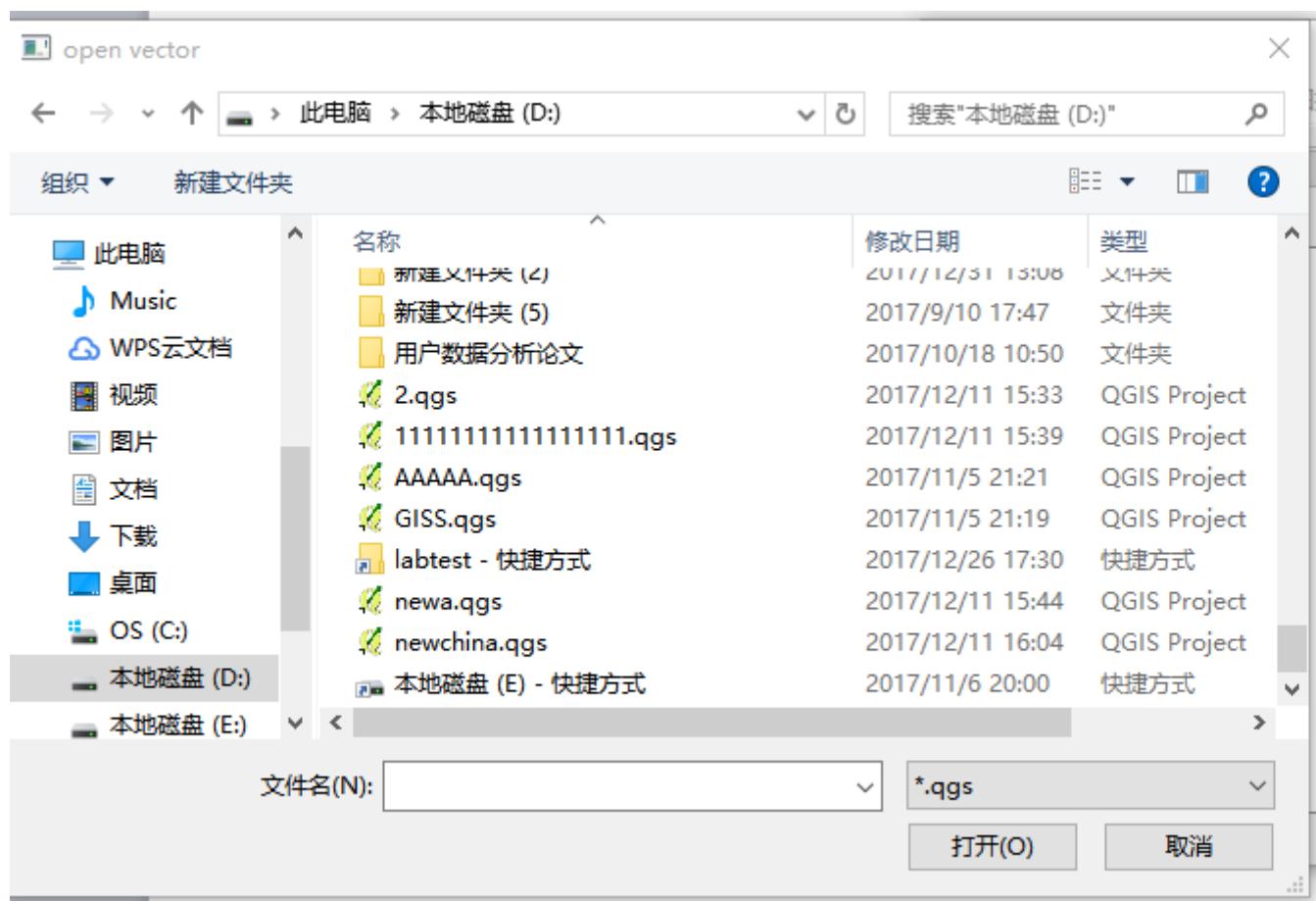
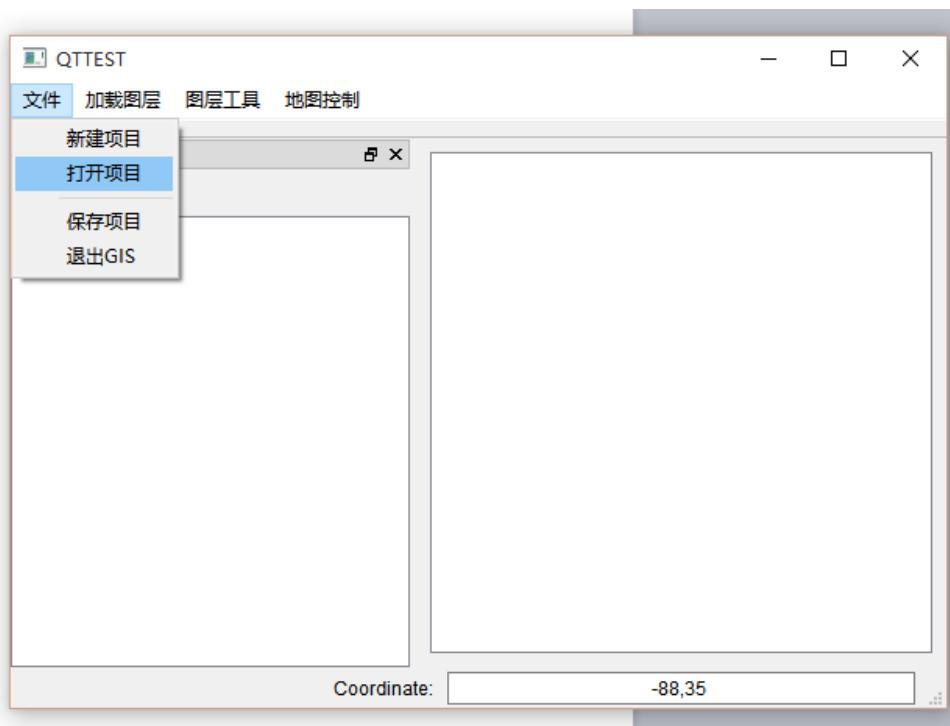
实现代码：

新建一个画布对象，将读取的工程文件的图层加载到新建的画布中。

filename 用于保存路径，调用 QgsProject 的 read()方法读取 qgs 项目。

```
//打开项目
void QTTEST::openfile()
{
    QgsMapCanvas * canvas = new QgsMapCanvas();
    QString filename = QFileDialog::getOpenFileName( this, tr( "open vector" ), "", "*.*" );
    QgsProject::instance()->read(QFileInfo(filename));
    QgsLayerTreeMapCanvasBridge * bridge = new QgsLayerTreeMapCanvasBridge(QgsProject::instance()->layerTreeRoot(), canvas);
}
```

实现效果：



在这里我打开了 newchina.qgs 项目。

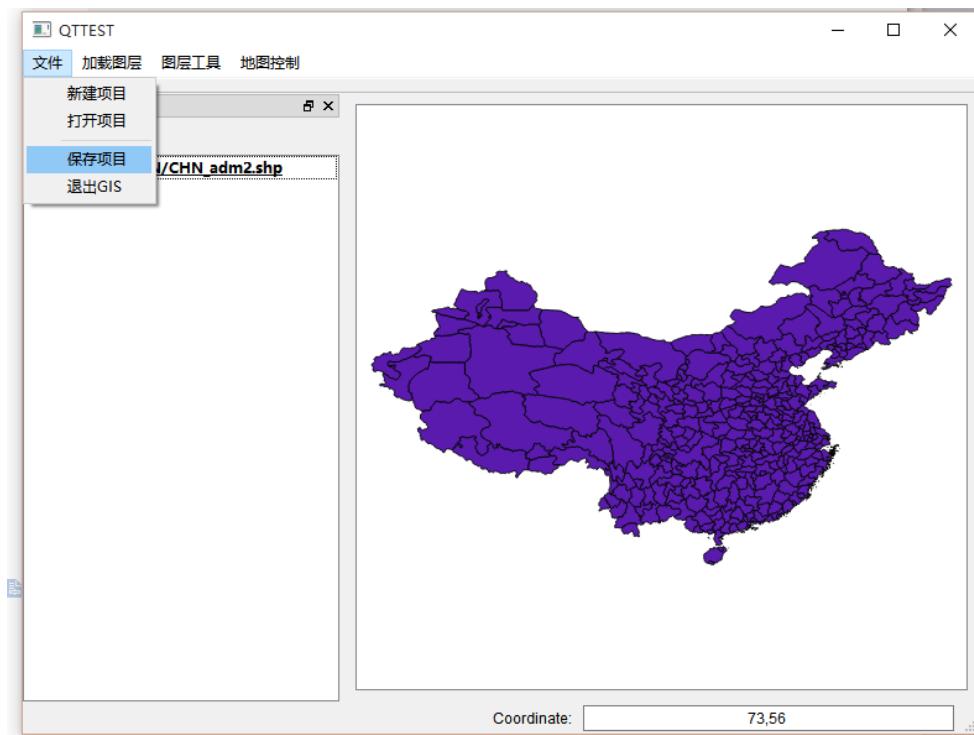
3.1.3 保存项目

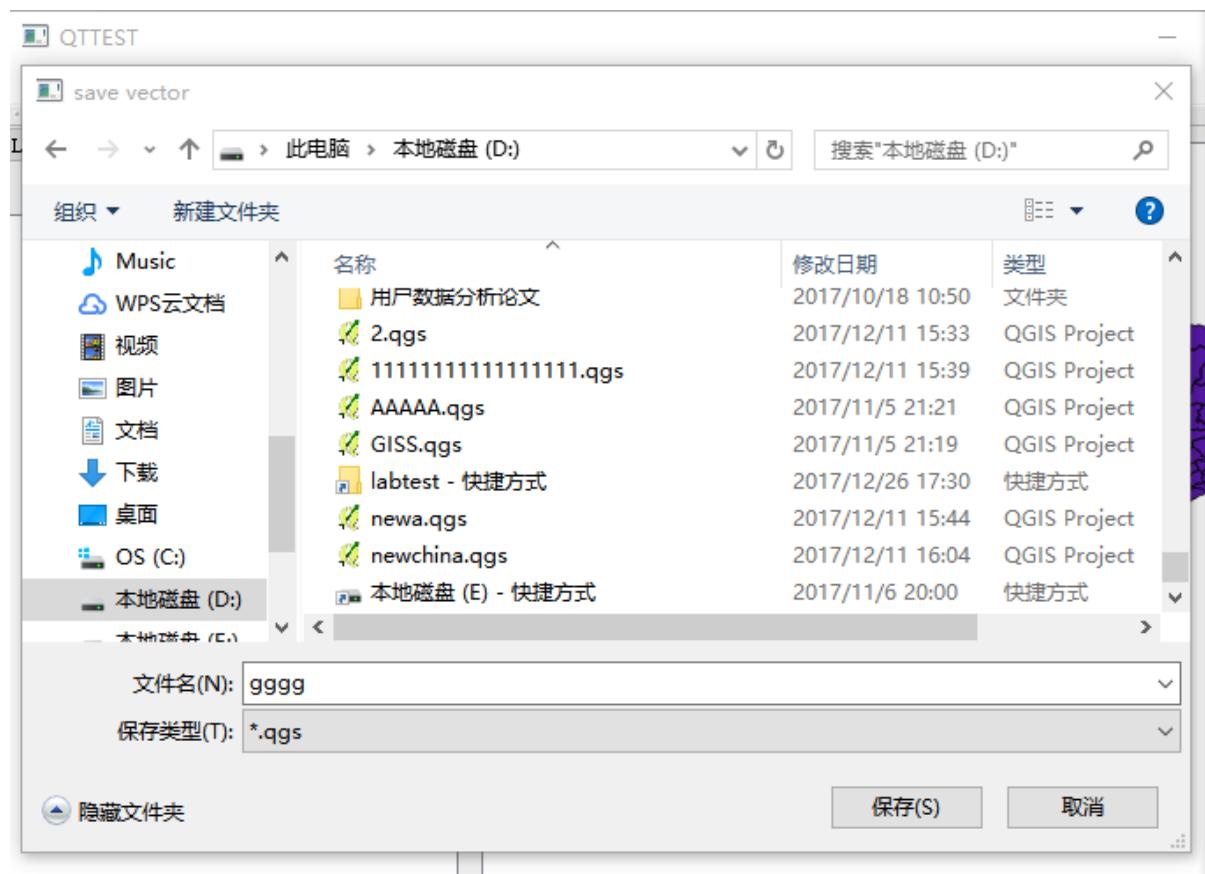
实现代码:

与打开 qgs 项目类似，通过 `getSaveFileName()` 函数得到文件路径名，再调用 `QgsProject` 的 `write` 方法将 qgs 项目保存到该路径。

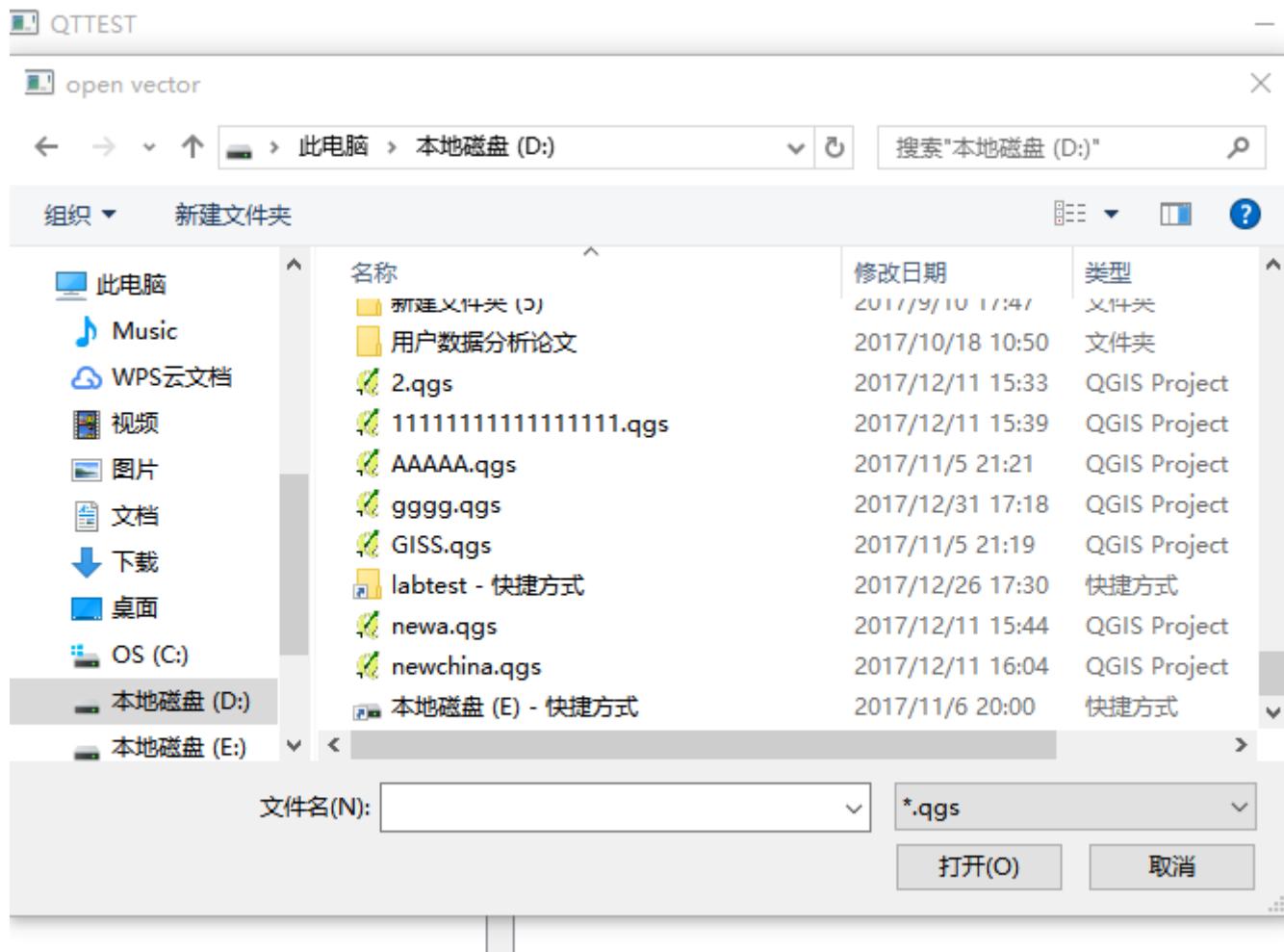
```
//保存项目
void QTTEST::savefile()
{
    QString filename = QFileDialog::getSaveFileName( this, tr( "save vector" ), "", "*.*" );
    QgsProject::instance()->write(QFileInfo(filename));
}
```

实现效果:





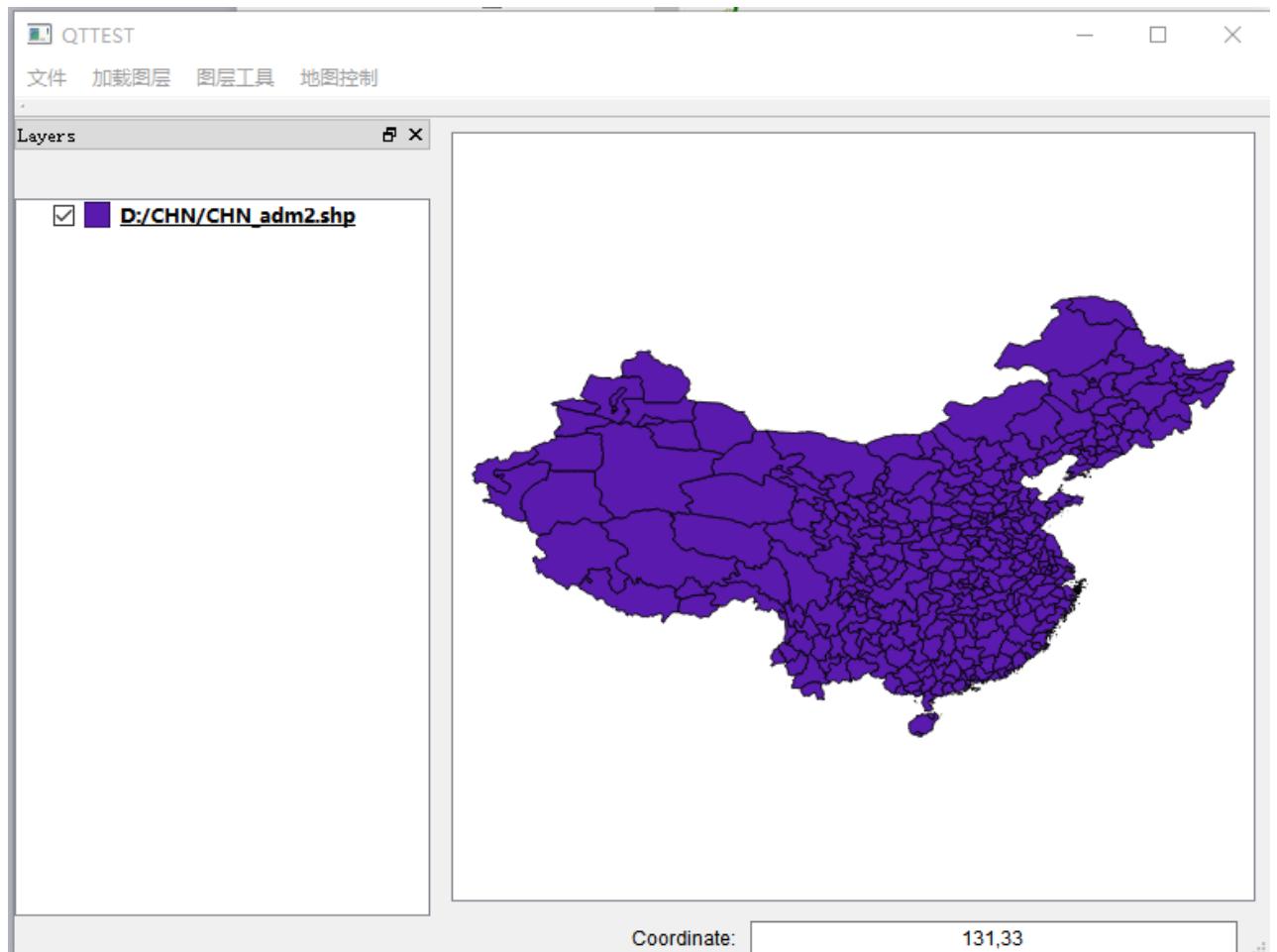
再点击打开文件，可以看到 gggg.qgs 已经保存到路径中。



3.1.4 关闭项目（退出 GIS）

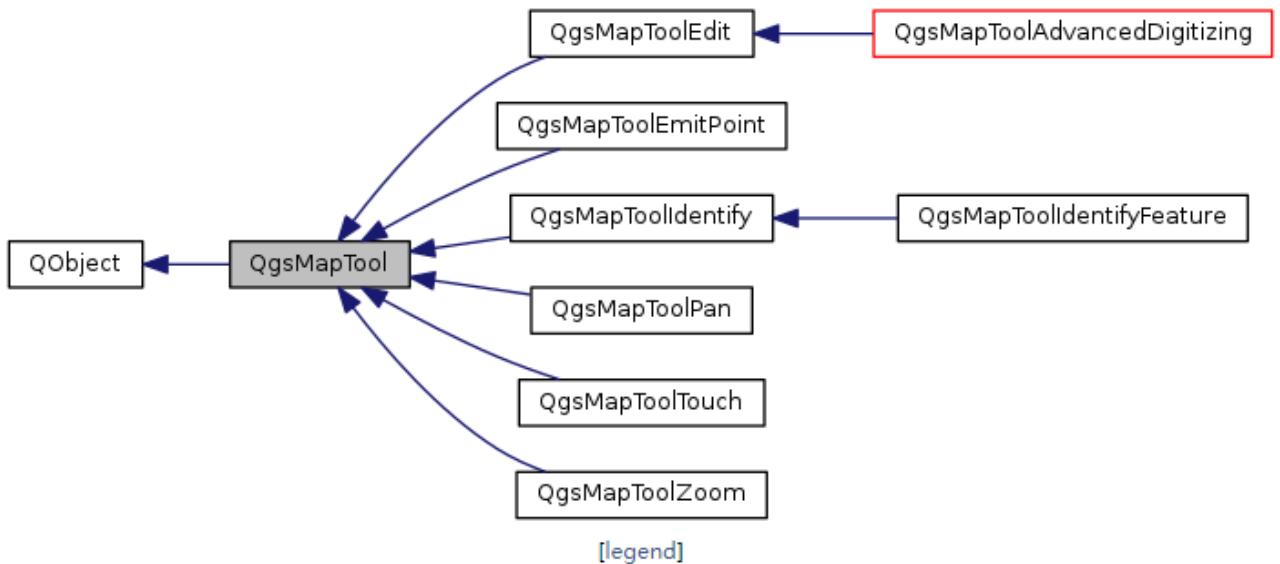
调用 QT 的 QApplication 类，类中的 quit() 函数可以退出 QT 程序，与点击右上角关闭效果类似。

```
//关闭GIS
void QTTEST::closefile()
{
    QApplication* app;
    app->quit();
}
```



3.2 地图操作

QGIS 中，对地图的操作在 QGIS 的 API 文件 QgsMapTool 中



其中 QgsMapToolPan 能够实现地图平移，QgsMapToolZoom 能够实现地图的放大缩小。调用相应的构造函数即可实现三个功能。再通过 QgsMapTool 的 setMapTool 函数调用相应的图层工具。

3.2.1 地图放大、缩小

实现代码：

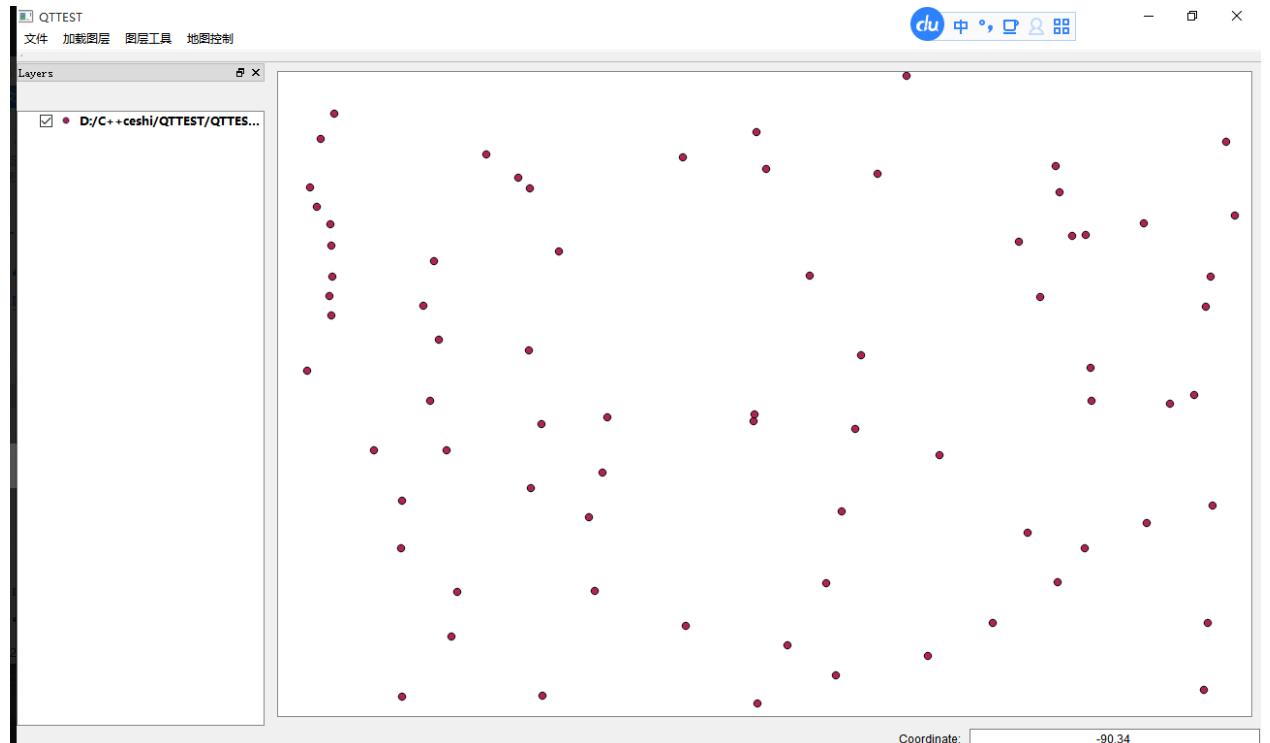
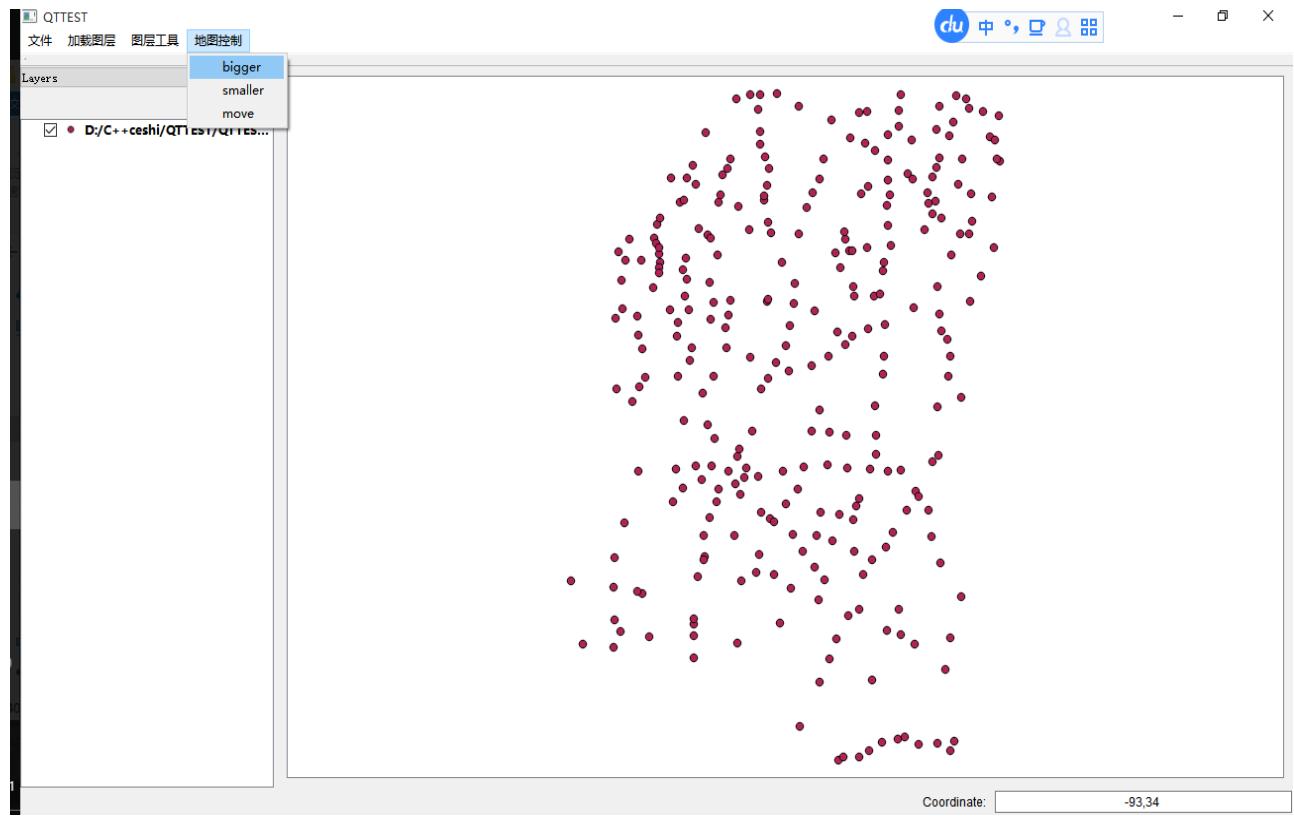
L,S 为 QgsMapTool* 类型，作为 QgsMapToolZoom 的构造函数参数，false 为放大功能，true 为缩小功能。

```
//*****
//地图放大、缩小、移动
//地图放大
void QTTEST::Large() {
    L=new QgsMapToolZoom(m_mapCanvas, false);
    m_mapCanvas->setMapTool(L);
}
//地图缩小
void QTTEST::Small() {
    S=new QgsMapToolZoom(m_mapCanvas, true);
    m_mapCanvas->setMapTool(S);
}
```

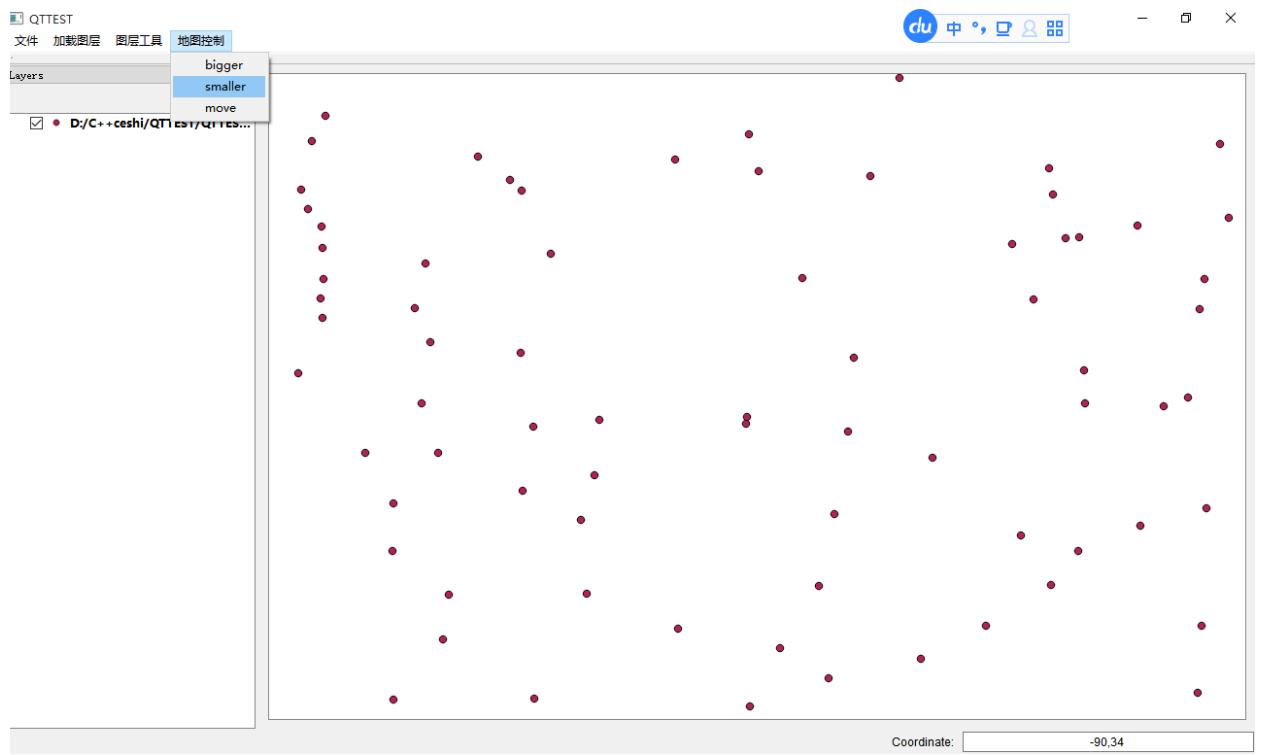
实现效果：

(点击相应功能，鼠标图像会变成放大镜+-哦)

放大



缩小



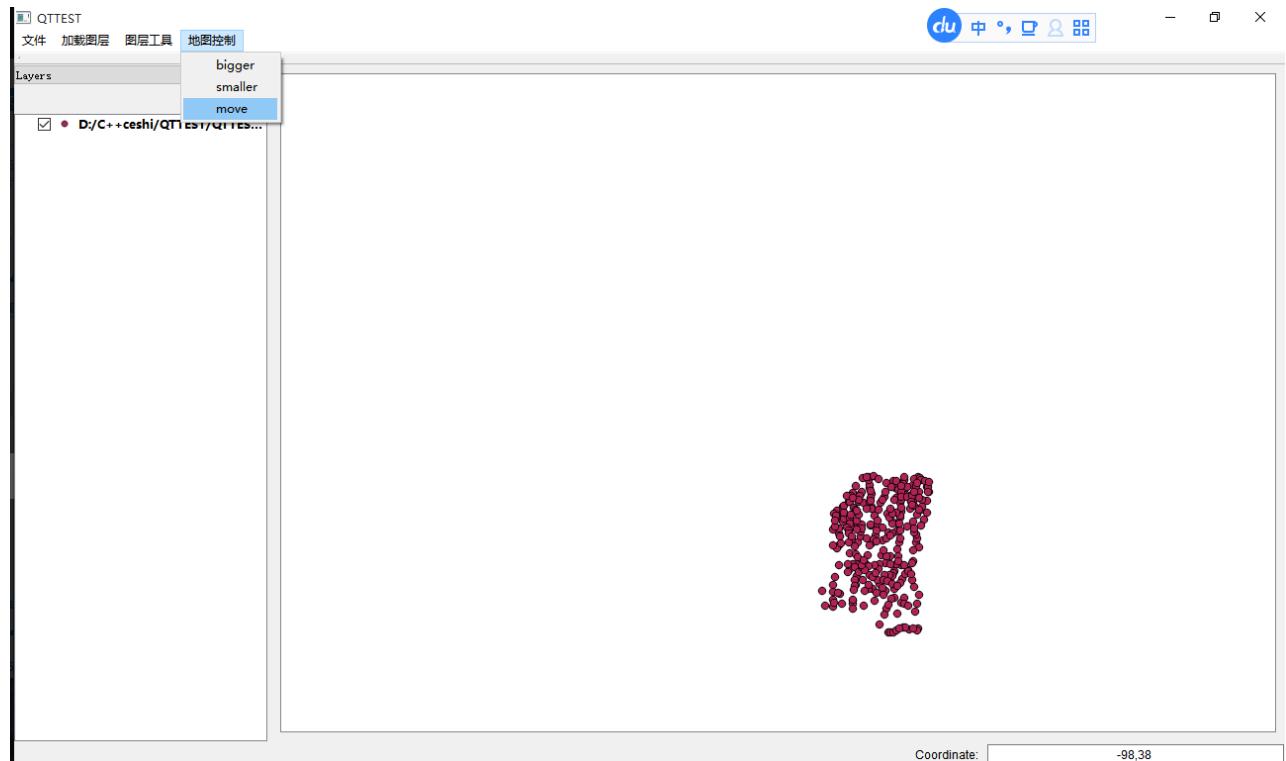
3.2.2 地图平移

实现代码：

M 依然是一个 QgsMapTool* 类型， QgsMapToolPan 构造函数能够实现平移功能。

```
//地图移动
void QTTEST::Move() {
    M=new QgsMapToolPan(m_mapCanvas);
    m_mapCanvas->setMapTool(M);
}
```

实现效果：



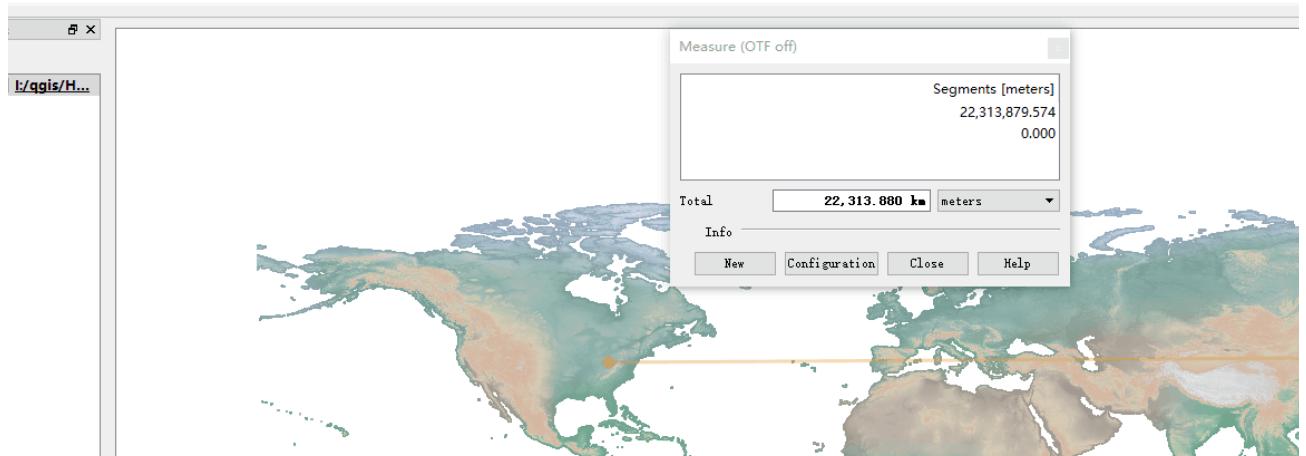


3.2.3 地图测距（替换为地图中鼠标坐标显示功能）

Qgis 的 app 目录中，有一个 qgsmeasuretool.h 函数，与放大缩小类似，该类继承自 QgsMapTool。新建一个 QgsMeasureTool 类的对象，再调用地图测距工具即可实现地图测距。可惜由于版本问题我的头文件无法正确的引入，在自己的电脑上试了很多次，依然无用，但我的这段代码在其他版本的 qgis 中是可以运行的。

```
//*****
//地图测距
void QTTEST::measure()
{
    //a=new QgsMeasureTool(m_mapCanvas, false);
    m_mapCanvas->setMapTool(a);
}
```

效果如图：



由于无法实现如此简单的地图测距功能！！！我将实现另一个功能：能够实时显示鼠标在地图中的坐标。

我认为如果能够得到地图中两个点的坐标，再计算两点距离。将两点距离换算为相应的比例尺，同样可以实现地图测距，只是计算的部分需要人为。

该方法来自于 **QgsProject** 类

实现代码：

函数传入 **QgsPoint** 引用类型变量，即为地图中某一点的对象。**QLineEdit*** **m_coordsEdit** 能够显示当前鼠标所在的坐标点的值，并且显示在底部状态栏中。

```
//*****
//获取当前鼠标所指地图的坐标
void QTTEST::showMouseCoordinate( const QgsPoint &p )
{
    if ( m_mapCanvas->mapUnits() == QGis::Degrees ) // 坐标度分秒的显示方式
    {
        QString format = QgsProject::instance()->readEntry( "PositionPrecision", "/DegreeFormat", "D" );
        if ( format == "DM" ) // 只显示度和分
        {
            m_coordsEdit->setText( p.toDegreesMinutes( m_MousePrecisionDecimalPlaces ) );
        }
        else if ( format == "DMS" ) // 显示度分秒
        {
            m_coordsEdit->setText( p.toDegreesMinutesSeconds( m_MousePrecisionDecimalPlaces ) );
        }
        else
        {
            m_coordsEdit->setText( p.toString( m_MousePrecisionDecimalPlaces ) );
        }
    }
    else
    {
        m_coordsEdit->setText( p.toString( m_MousePrecisionDecimalPlaces ) );
    }

    if ( m_coordsEdit->width() > m_coordsEdit->minimumWidth() )
    {
        m_coordsEdit->setMinimumWidth( m_coordsEdit->width() );
    }
}
```

在主函数中利用 connect，以鼠标位置 xyCoordinates (const QgsPoint&) 为触发器。实时显示坐标点值。

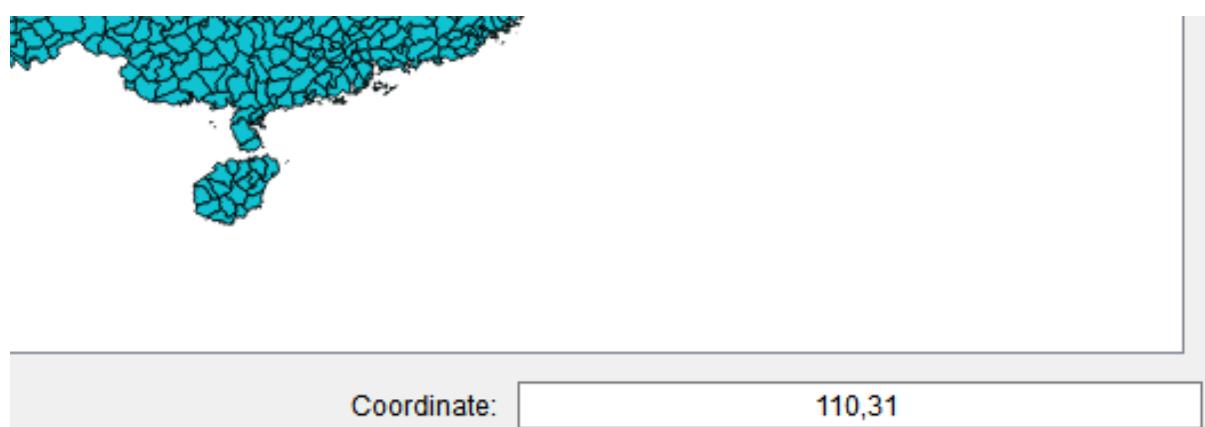
```
connect( m_mapCanvas, SIGNAL( xyCoordinates( const QgsPoint& ) ), this, SLOT( showMouseCoordinate( const QgsPoint& ) ) );
```

实现效果：

由于截屏无法显示鼠标，在这里说明一下，现在我的鼠标指向了地图中武汉的位置。



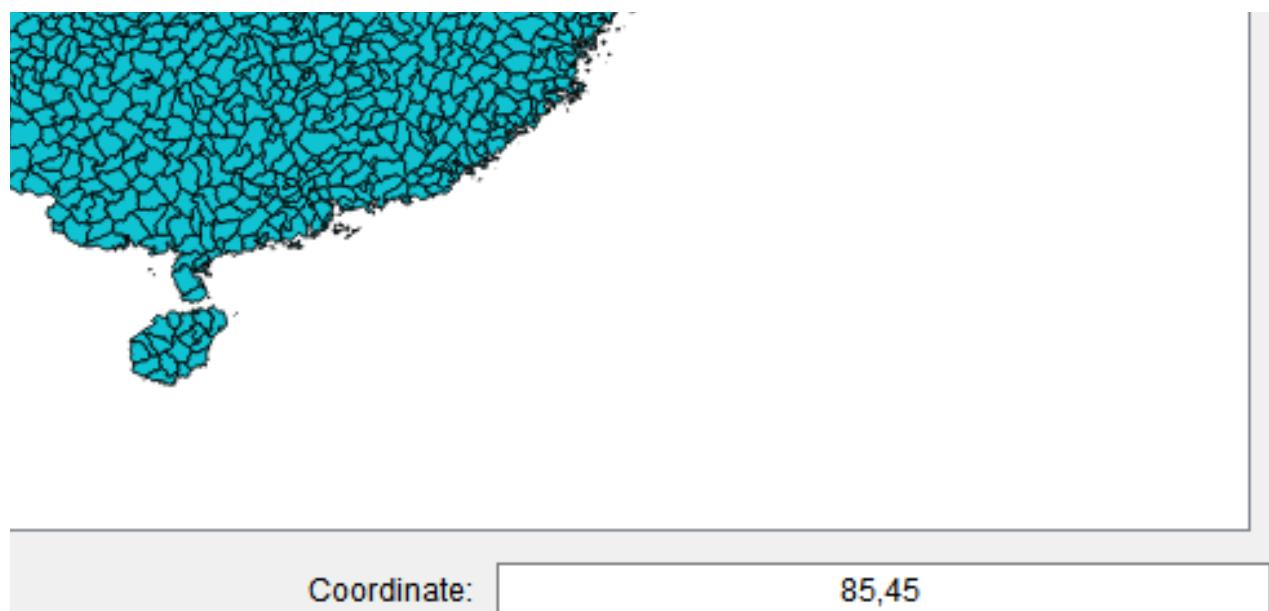
显示坐标为 (110.31)



我又将鼠标指向大西北的某个位置



得到的坐标值为 (85,45)



得到了两个坐标点之后就可以进行相应的距离计算，再将得到的距离换算后，便可测得两个点的实际距离。

附加功能： 显示鼠标坐标的底部的状态栏实现，效果如上图。

实现代码：

```
void QTTEST::createStatusBar()
{
    statusBar()->setStyleSheet( "QStatusBar::item {border: none;}" );

    QFont myFont( "Arial", 9 );
    statusBar()->setFont( myFont );

    //添加坐标显示标签
    m_coordsLabel = new QLabel( QString(), statusBar() );
    m_coordsLabel->setObjectName( "m_coordsLabel" );
    m_coordsLabel->setFont( myFont );
    m_coordsLabel->setMinimumWidth( 10 );
    m_coordsLabel->setMargin( 3 );
    m_coordsLabel->setAlignment( Qt::AlignCenter );
    m_coordsLabel->setFrameStyle( QFrame::NoFrame );
    m_coordsLabel->setText( tr( "Coordinate:" ) );
    m_coordsLabel->setToolTip( tr( "Current map coordinate" ) );
    statusBar()->addPermanentWidget( m_coordsLabel, 0 );

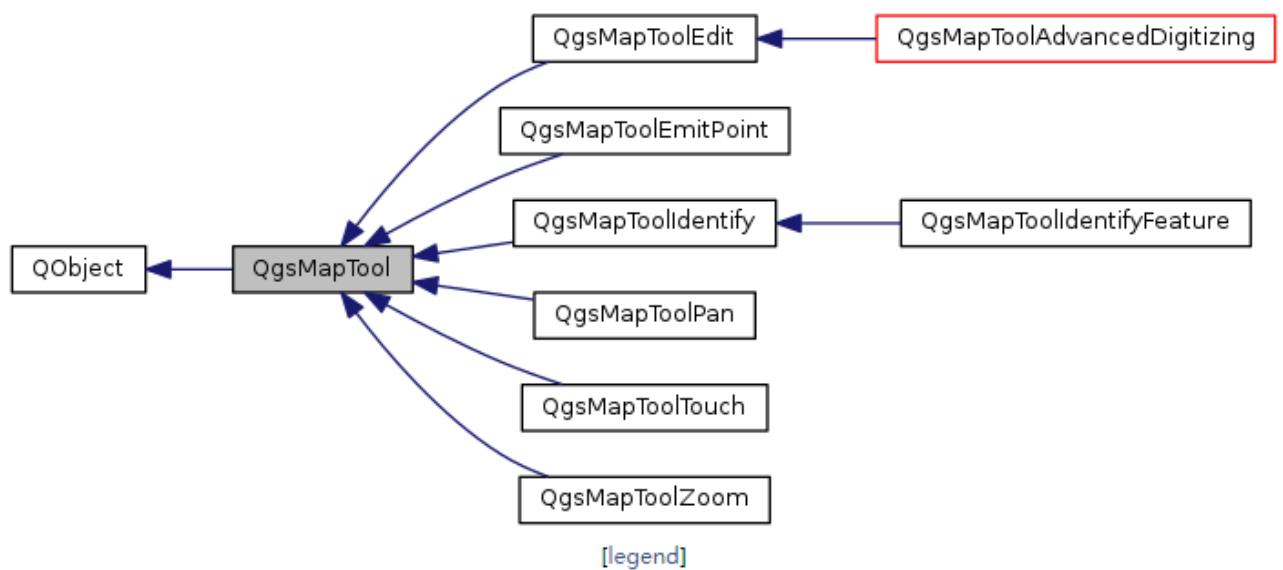
    //坐标编辑标签
    m_coordsEdit = new QLineEdit( QString(), statusBar() );
    m_coordsEdit->setObjectName( "m_coordsEdit" );
    m_coordsEdit->setFont( myFont );
    m_coordsEdit->setMinimumWidth( 10 );
    m_coordsEdit->setMaximumWidth( 300 );
    m_coordsEdit->setContentsMargins( 0, 0, 0, 0 );
    m_coordsEdit->setAlignment( Qt::AlignCenter );
    statusBar()->addPermanentWidget( m_coordsEdit, 0 );
    m_coordsEdit->setReadOnly( true );//只读
    connect( m_coordsEdit, SIGNAL( returnPressed() ), this, SLOT( userCenter() ) );

    m_dizzyTimer = new QTimer( this );
    connect( m_dizzyTimer, SIGNAL( timeout() ), this, SLOT( dizzy() ) );
}
```

3.3 图层管理

3.3.1 图元识别

图元识别是指能够识别鼠标选中的地图中图元所拥有的属性值。该功能的实现依然在 QGIS 的 API 的 QgsMapTool 类中。其中 QgsMapToolIdentify 类便是能够实现属性识别功能的类。



由于该类中，响应鼠标事件的函数没有相应的响应动作。

```
void QgsMapToolIdentify::canvasMoveEvent( QMouseEvent * e )
{
    Q_UNUSED( e );
}

void QgsMapToolIdentify::canvasPressEvent( QMouseEvent * e )
{
    Q_UNUSED( e );
}

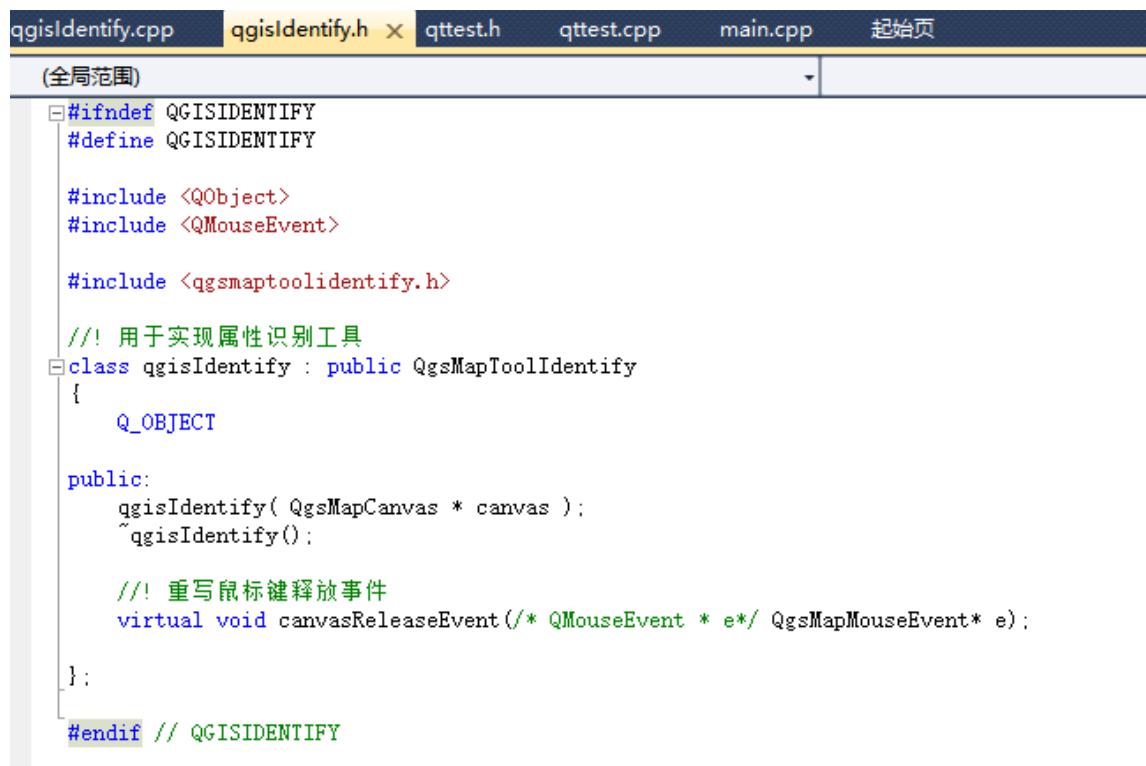
void QgsMapToolIdentify::canvasReleaseEvent( QMouseEvent * e )
{
    Q_UNUSED( e );
}
```

所以如果直接调用该函数，并不会实现图元识别功能。因此需要自己定义一个识别类，并重写鼠标释放事件函数，才能够实现真正的图元识别。

实现代码：

我新建了一个 CPP 与它的头文件文件 ，用于写自定义的图元识别类。

头文件代码：将鼠标释放函数设置为虚函数便于重写该方法。在这里要注意一个问题，就是我想实现图元识别的功能，这个图元是 QGIS 中的图层，事件发生在图层上，而不仅仅是 QT 的面板中，所以要响应相应的鼠标释放事件，类型必须是 QgsMapMouseEvent 类型的，不可以是 QT 的 QMouseEvent，这样才能得到正确图层的点击释放事件。



```
qgisIdentify.cpp  qgisIdentify.h  qttest.h  qttest.cpp  main.cpp  起始页
(全局范围)
#ifndef QGISIDENTIFY
#define QGISIDENTIFY

#include <QObject>
#include <QMouseEvent>

#include <qgsmaptoolidentify.h>

//! 用于实现属性识别工具
class qgisIdentify : public QgsMapToolIdentify
{
    Q_OBJECT

public:
    qgisIdentify( QgsMapCanvas * canvas );
    ~qgisIdentify();

    //! 重写鼠标键释放事件
    virtual void canvasReleaseEvent( /* QMouseEvent * e */ QgsMapMouseEvent* e );

};

#endif // QGISIDENTIFY
```

重写后的鼠标释放事件方法：

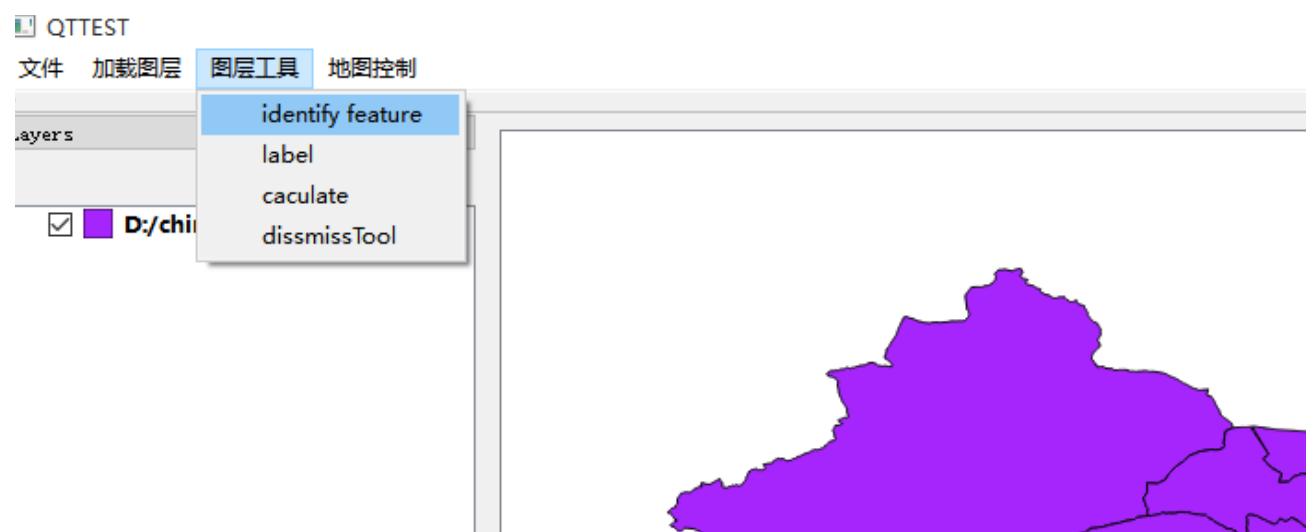
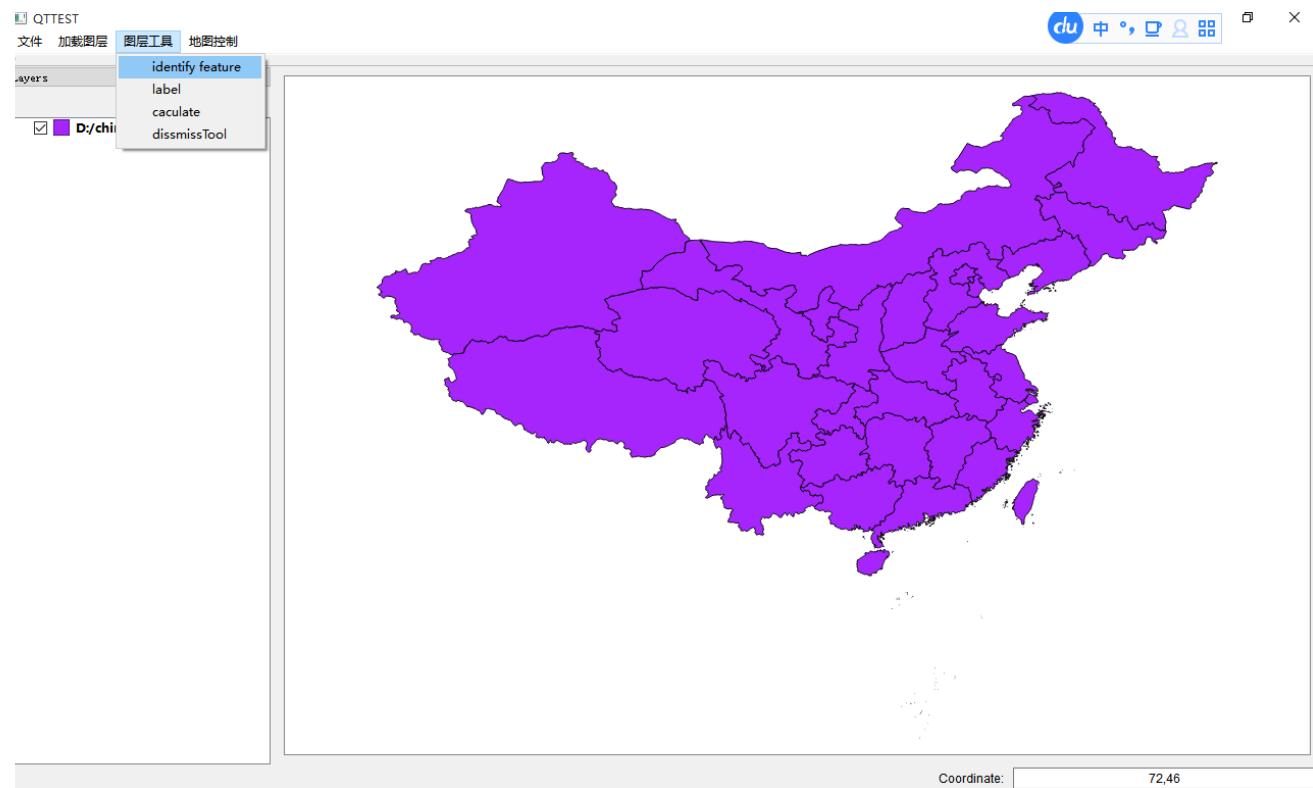
```
void qgisIdentify::canvasReleaseEvent( /*QMouseEvent * e*/ QgsMapMouseEvent *e )
{
    IdentifyMode mode = QgsMapToolIdentify::LayerSelection; // 控制识别模式
    QList<IdentifyResult> results = QgsMapToolIdentify::identify( e->x(), e->y(), mode ); // 这句返回识别结果

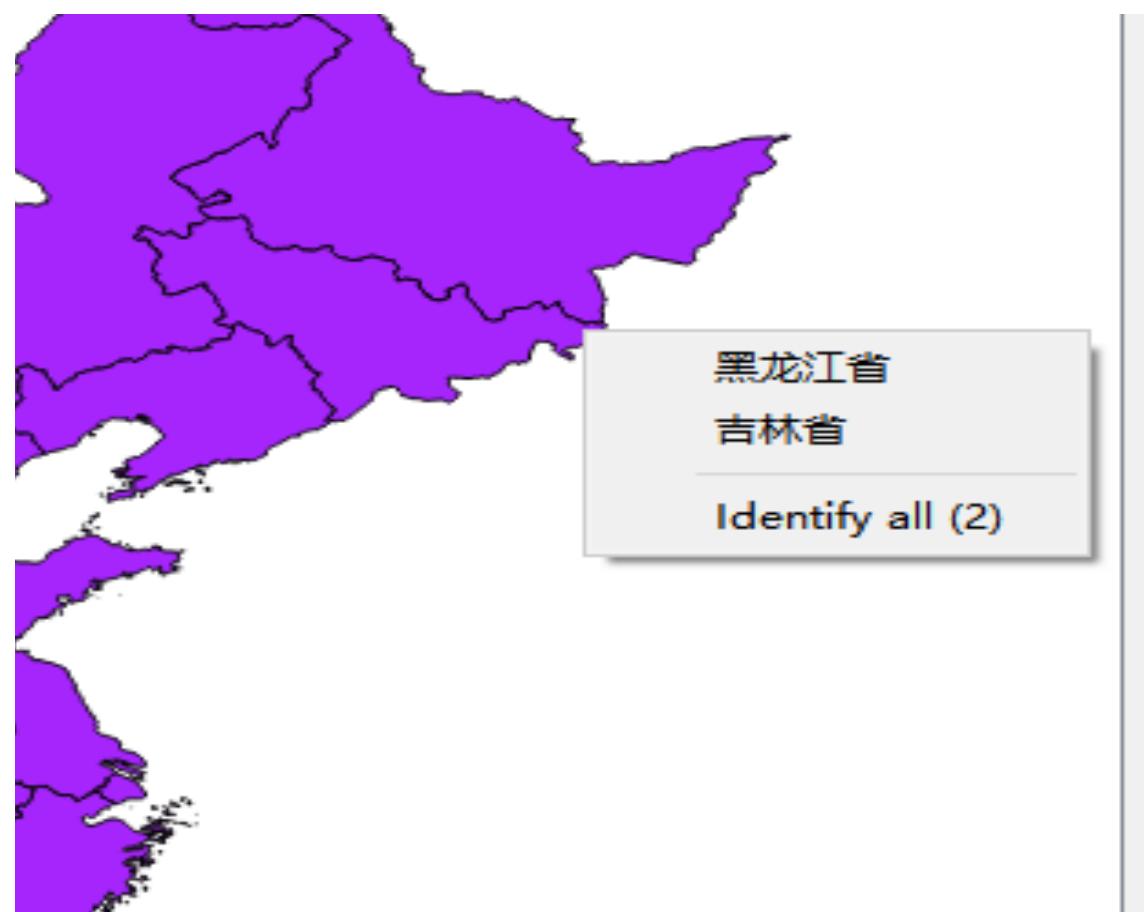
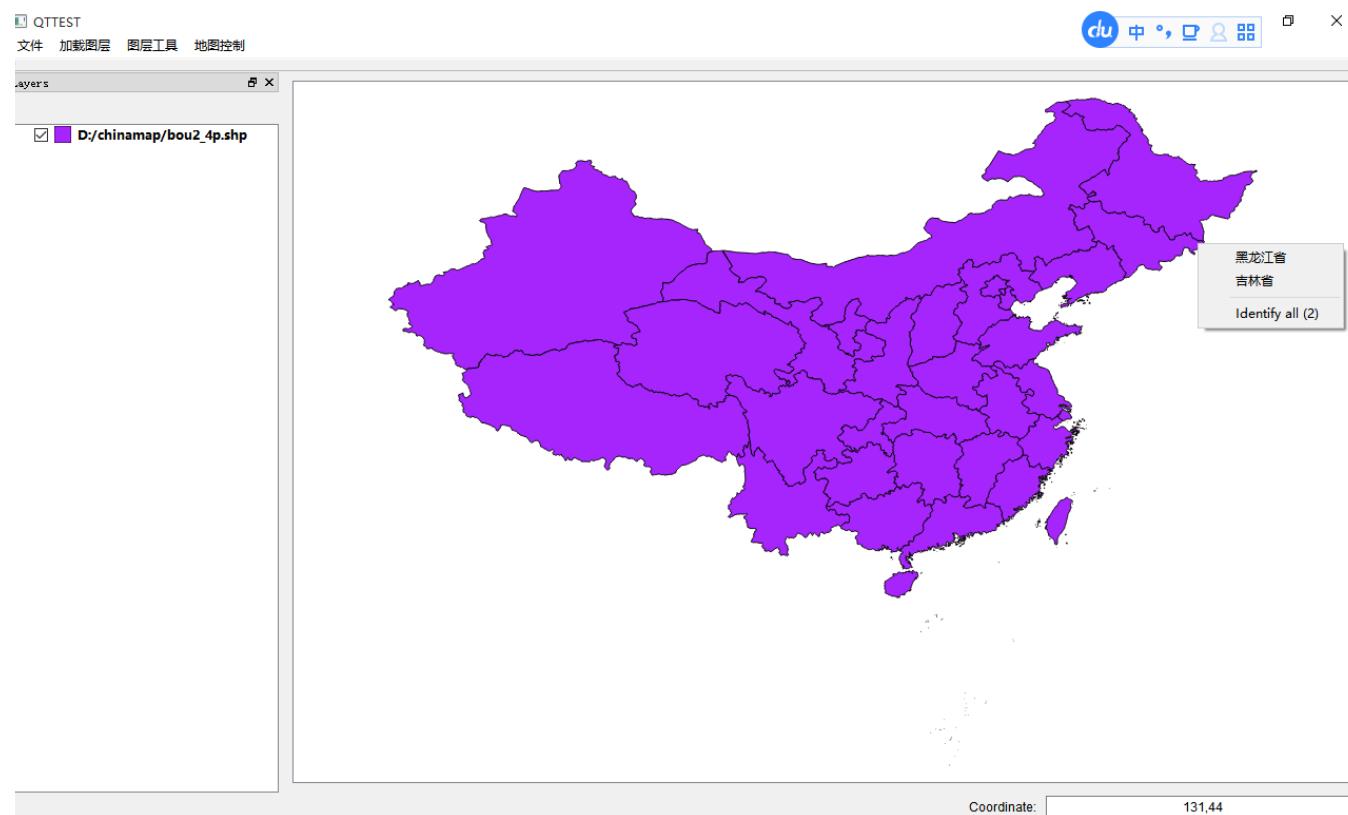
    if ( results.isEmpty() )
    {
        QTTEST::instance()->statusBar()->showMessage( tr( "No features at this position found." ) );
    }
    else
    {
        // 显示出识别结果，这里仅作示例，结果的展示方式可以自定义
        IdentifyResult feature = results.at( 0 );
        QString title = feature.mLayer->name();
        QString content = feature.mFeature.attribute( 1 ).toString();
        // 显示识别窗口
        QMessageBox::critical( NULL,
                              title,
                              content );
    }
}
```

在函数调用时新建该类的构造函数并激活，再调用识别工具即可。

```
//*****
//图元识别
void QTTEST::qIdentify()
{
    m_mapIdentify=new qgisIdentify(m_mapCanvas);
    m_mapIdentify->activate();
    m_mapCanvas->setMapTool(m_mapIdentify);
}
```

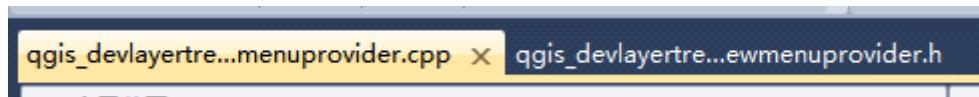
实现效果：





附加功能：

在实现添加图层之前，首先要实现图层管理器的功能，在如下文件中。



其中重写 createContextMenu()方法能够获得右键菜单功能。

```
QMenu* qgis_devLayerTreeViewMenuProvider::createContextMenu()
{
    // 设置这个路径是为了获取图标文件
    QString iconDir = "../images/themes/default/";

    QMenu* menu = new QMenu;

    QgsLayerTreeViewDefaultActions* actions = mView->defaultActions();

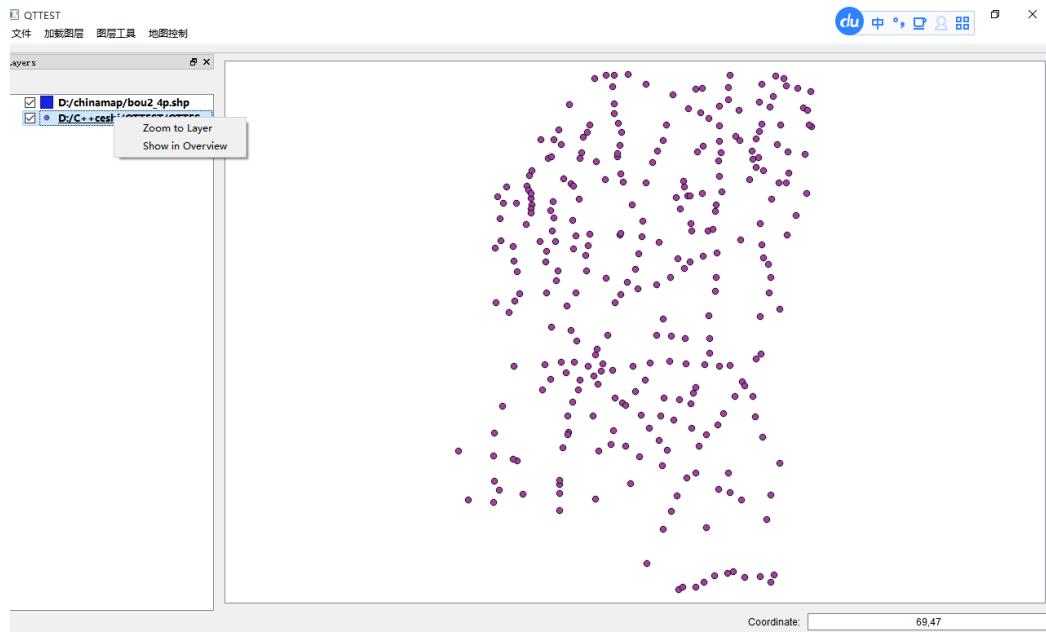
    QModelIndex idx = mView->currentIndex();

    // global menu
    if ( !idx.isValid() )
    {
        menu->addAction( actions->actionAddGroup( menu ) );
        menu->addAction( QIcon( iconDir + "mActionExpandTree.png" ), tr( "&Expand All" ), mView, SLOT( expandAll() ) );
        menu->addAction( QIcon( iconDir + "mActionCollapseTree.png" ), tr( "&Collapse All" ), mView, SLOT( collapseAll() ) );
        menu->addAction( QIcon( iconDir + "mActionRemoveLayer.svg" ), tr( "&Remove" ), QTTEST::instance(), SLOT( removeLayer() ) );
    }
    else if ( QgsLayerTreeNode* node = mView->layerTreeModel()->index2node( idx ) )
    {
        // layer or group selected
        if ( QgsLayerTree::isGroup( node ) )
        {
            menu->addAction( actions->actionZoomToGroup( mCanvas, menu ) );
            menu->addAction( QIcon( iconDir + "mActionRemoveLayer.svg" ), tr( "&Remove" ), QTTEST::instance(), SLOT( removeLayer() ) );
            menu->addAction( actions->actionRenameGroupOrLayer( menu ) );
            if ( mView->selectedNodes( true ).count() >= 2 )
            {
                menu->addAction( actions->actionGroupSelected( menu ) );
            }
            menu->addAction( actions->actionAddGroup( menu ) );
        }
        else if ( QgsLayerTree::isLayer( node ) )
        {
            QgsMapLayer* layer = QgsLayerTree::toLayer( node )->layer();
            menu->addAction( actions->actionZoomToLayer( mCanvas, menu ) );
            menu->addAction( actions->actionShowInOverview( menu ) );

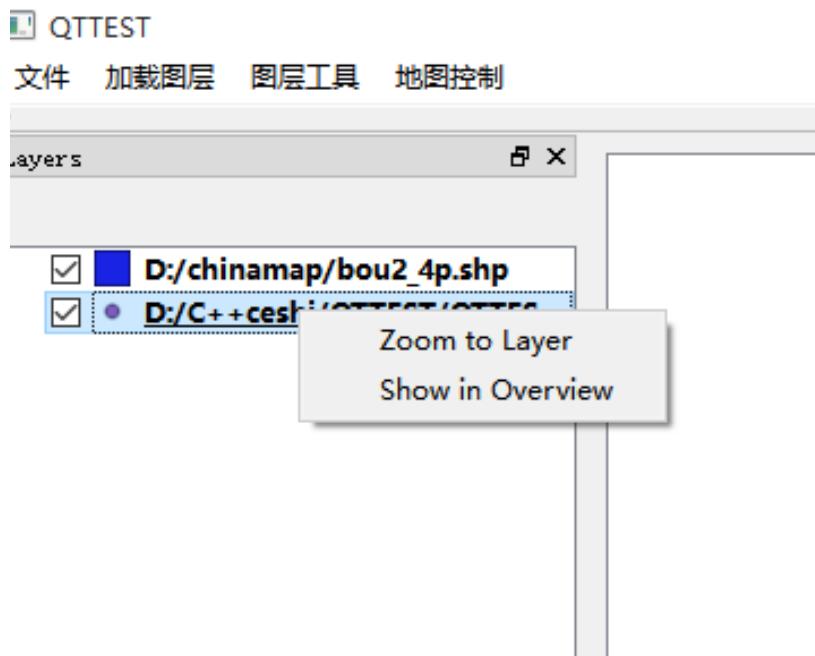
            // 如果选择的是栅格图层
            if ( layer && layer->type() == QgsMapLayer::RasterLayer )
            {
                // TO DO
            }
        }
    }

    return menu;
}
```

实现效果：

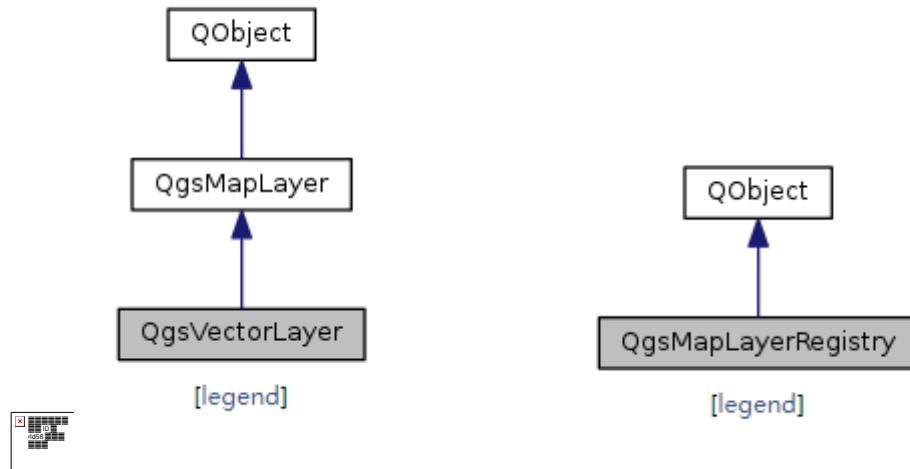


Zoom to Layer 实现了将选中的图层置于最上层显示的功能，Show in Overview 我还没有实现。



3.3.2 添加图层

在 QGIS 的 API 中需要用到如下两个类。



(1) 添加矢量地图

实现代码：

Filename, basename, temp 用于保存路径名等内容。

QgsVectorLayer 的构造函数传入图层路径，与支持的插件名称后能够得到相应的图层。再调用 QgsMapLayerRegistry 类注册图层，m_mapCanvas 对象调用了几个函数将图层展开显示。

```
/* **** */
//添加各种图层
//Add_VectorLayers
void QTTEST::addVectorLayers()
{
    QString filename = QFileDialog::getOpenFileName( this, tr( "open vector" ), "", "* .shp" );
    QStringList temp = filename.split( QDir::separator() );
    QString basename = temp.at( temp.size() - 1 );
    QgsVectorLayer* vecLayer = new QgsVectorLayer(filename, basename, "ogr", true );
    //QMessageBox::critical(this, "error", filename);
    //QMessageBox::critical(this, "error", basename);
    if(!vecLayer->isValid()){

        QMessageBox::critical(this, "error", "layer is invalid");
        return;
    }

    QgsMapLayerRegistry::instance()->addMapLayer( vecLayer );
    mapCanvasLayerSet.append( vecLayer );
    m_mapCanvas->setExtent( vecLayer->extent() );
    m_mapCanvas->setLayerSet( mapCanvasLayerSet );
    m_mapCanvas->setVisible( true );
    m_mapCanvas->freeze( false );
    m_mapCanvas->refresh();

    ///////////////////////////////////////////////////////////////////20171227 update
    //vecLayer->enableLabels(true); //显示图层标注
    //m_mapCanvas->refresh();
    ///////////////////////////////////////////////////////////////////end update
}

```

实现效果：

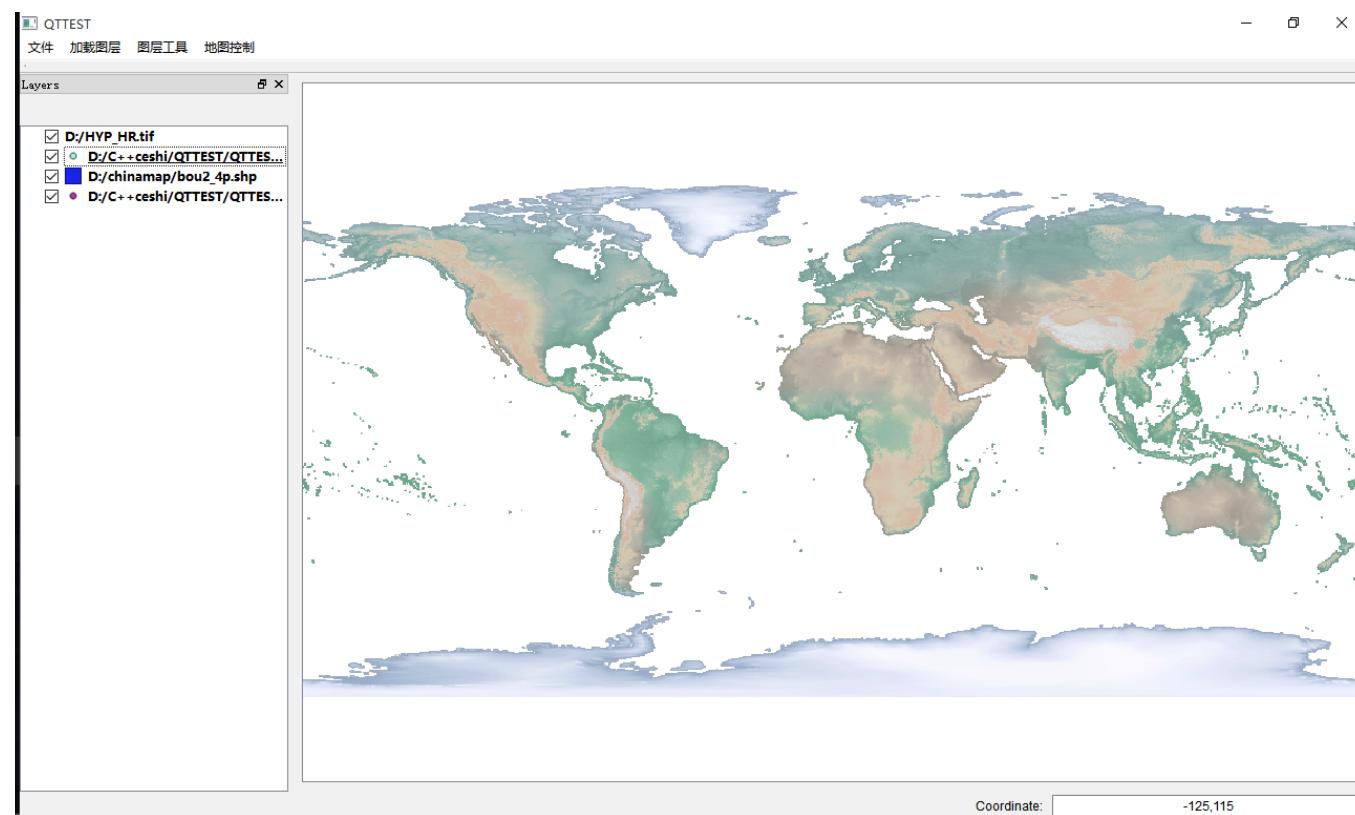
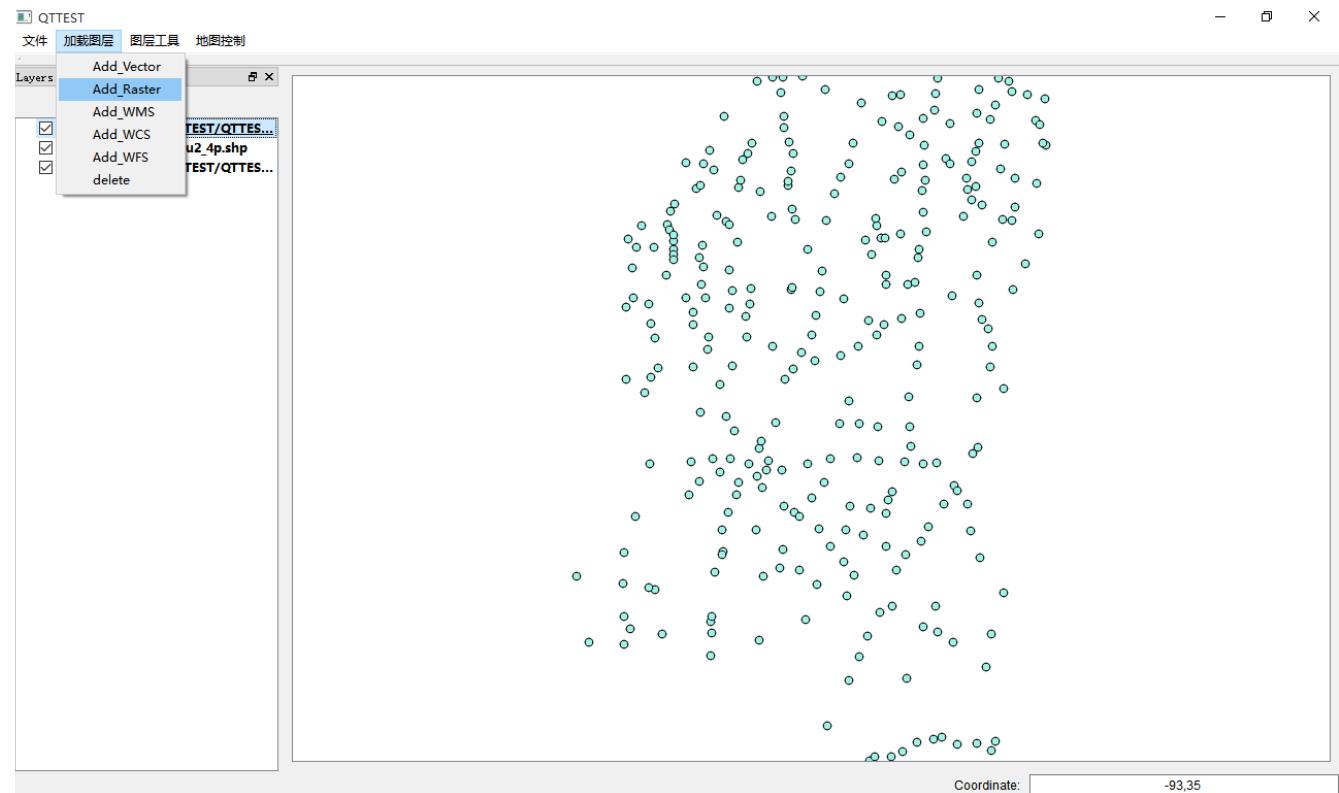


(2) 添加影像地图

实现代码: 与矢量地图类似，只是需要改一下插件名称。

```
r( filename, basename, "gdal", false );
```

实现效果:



(3) 添加 WFS、WCS、WMS 图层

WMS、WCS（与 WFS 类似）实现代码：

```
//ADD WMS
void QTTEST::addWMSLayers()
{
    QDialog *wms = dynamic_cast<QDialog*>(QgsProviderRegistry::instance()->selectWidget( QString("WMS")));
    if ( !wms )
    {
        statusBar()->showMessage( tr( "cannot add wms layer." ), 10 );
    }

    connect( wms, SIGNAL( addRasterLayer( QString const &, QString const &, QString const & ) ),
             this, SLOT( addOpenSourceRasterLayer( QString const &, QString const &, QString const & ) ) );
    wms->exec();

    delete wms;
}
```

需要注意的问题是，要在 main 函数中添加如下代码，才可以 connection 成功。

```
QCoreApplication::setOrganizationName( "oneone" );
QCoreApplication::setOrganizationDomain( "oneone.com" ); // 域名好像是可以不用加的
QCoreApplication::setApplicationName( "QTTEST" );
```

WFS 实现代码:

```
//ADD WFS
void QTTEST::addWFSLayers()
{
    if ( !m_mapCanvas ) {return;}

    QDialog *wfs = dynamic_cast<QDialog*>( QgsProviderRegistry::instance()->selectWidget( QString( "WFS" ), + );
    if ( !wfs )
    {
        QMessageBox::warning( this, tr( "WFS" ), tr( "Cannot get WFS select dialog from provider." ) );
        return;
    }
    connect( wfs, SIGNAL( addWfsLayer( QString, QString ) ),
             this, SLOT( addWfsLayer( const QString, const QString ) ) );

    //re-enable wfs with extent setting: pass canvas info to source select
    wfs->setProperty( "MapExtent", m_mapCanvas->extent().toString() );
    if ( m_mapCanvas->mapSettings().hasCrsTransformEnabled() )
    {
        //if "on the fly" reprojection is active, pass canvas CRS
        wfs->setProperty( "MapCRS", m_mapCanvas->mapSettings().destinationCrs().authid() );
    }

    bool bkRenderFlag = m_mapCanvas->renderFlag();
    m_mapCanvas->setRenderFlag( false );
    wfs->exec();
    m_mapCanvas->setRenderFlag( bkRenderFlag );
    delete wfs;
}
```

需要添加矢量图层:

```
// 添加矢量图层
void QTTEST::addWFSLayer( const QString& url, const QString& typeName )
{
    QgsVectorLayer* vecLayer = new QgsVectorLayer( url, typeName, "WFS", false );
    if ( !vecLayer->isValid() )
    {
        QMessageBox::critical( this, "error", "layer is invalid" );
        return;
    }

    QgsMapLayerRegistry::instance()->addMapLayer( vecLayer );
    mapCanvasLayerSet.append( vecLayer );
    m_mapCanvas->setExtent( vecLayer->extent() );
    m_mapCanvas->setLayerSet( mapCanvasLayerSet );
    m_mapCanvas->setVisible( true );
    m_mapCanvas->freeze( false );
    m_mapCanvas->refresh();
}

void QTTEST::addOpenSourceRasterLayer( const QString& url, const QString& basename, const QString& providerKey )
{
    QgsRasterLayer *rasterLayer = 0;

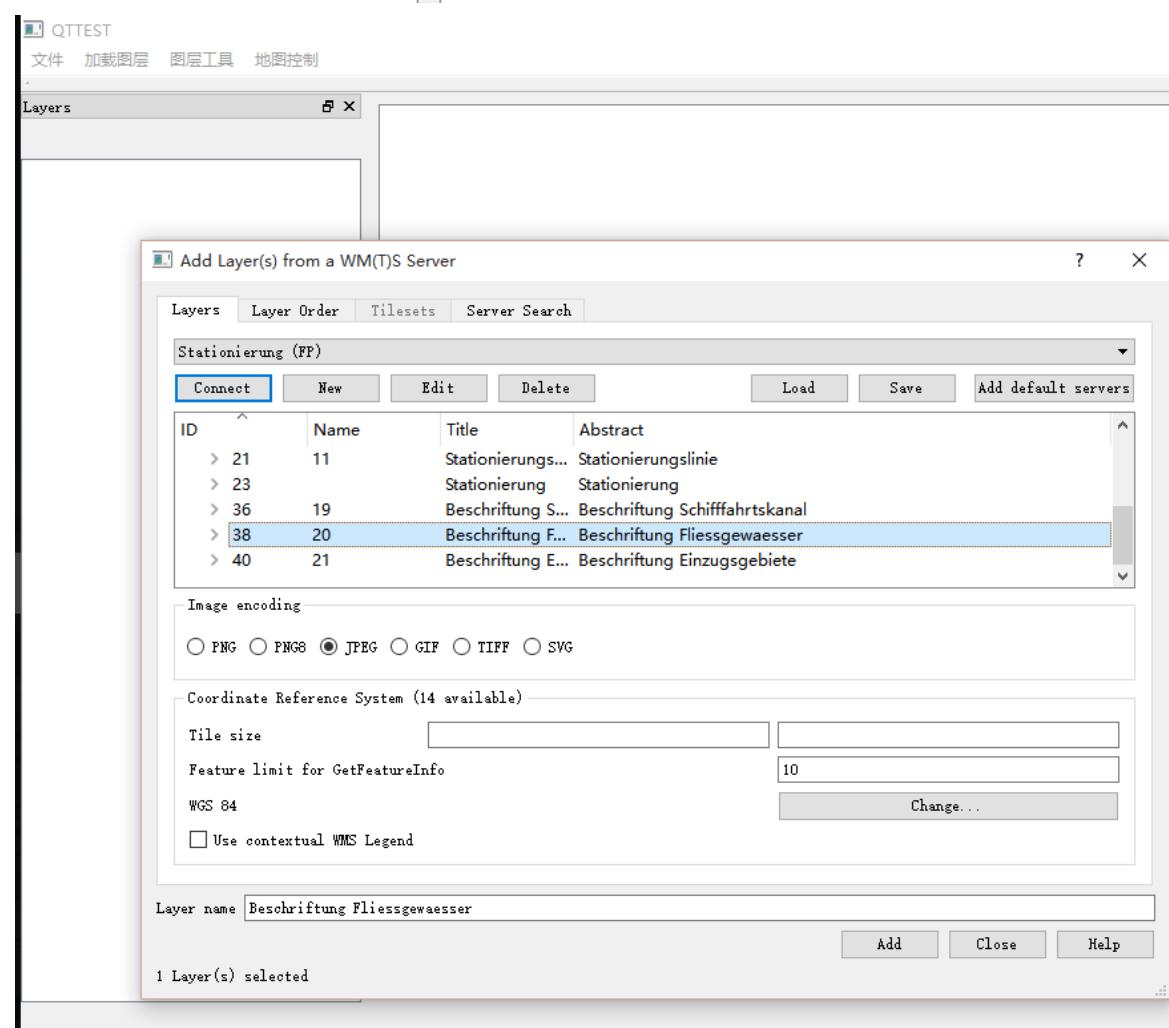
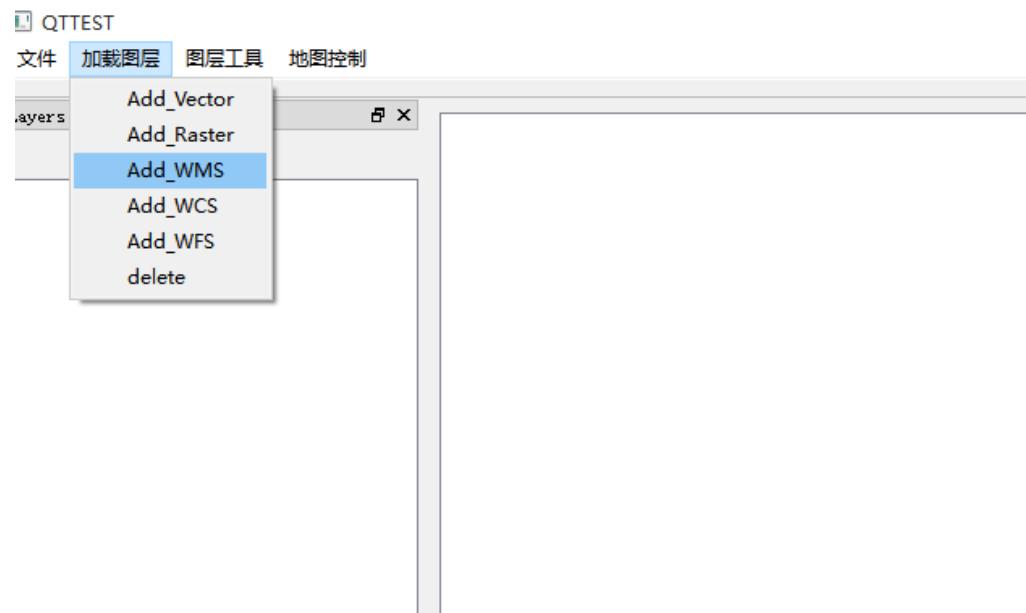
    if ( providerKey.isEmpty() )
    {
        rasterLayer = new QgsRasterLayer( url, basename );
    }
    else
    {
        rasterLayer = new QgsRasterLayer( url, basename, providerKey );
    }

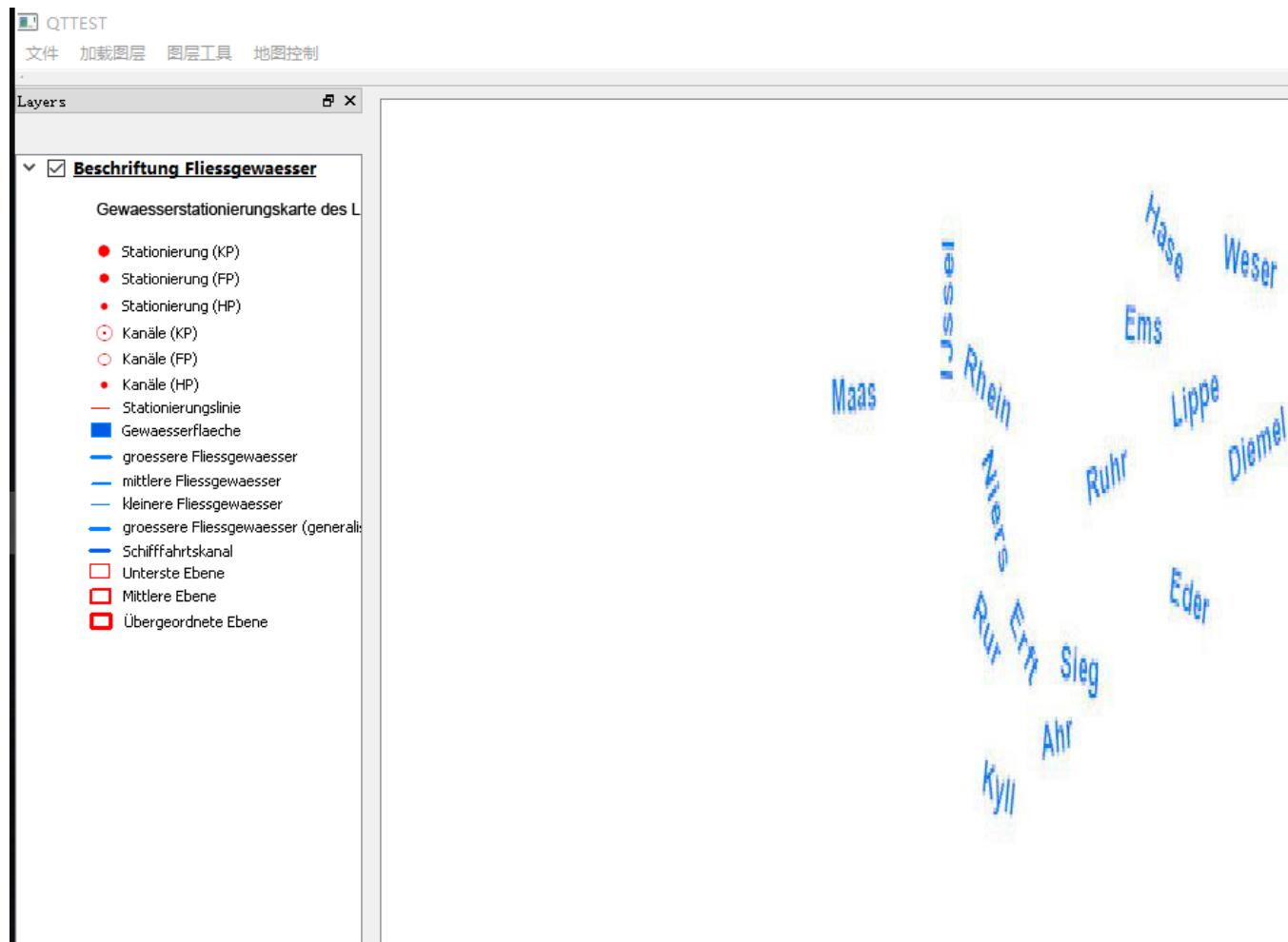
    if ( !rasterLayer->isValid() )
    {
        QMessageBox::critical( this, "error", "layer is invalid" );
        return;
    }

    QgsMapLayerRegistry::instance()->addMapLayer( rasterLayer );
    mapCanvasLayerSet.append( rasterLayer );
    m_mapCanvas->setExtent( rasterLayer->extent() );
    m_mapCanvas->setLayerSet( mapCanvasLayerSet );
    m_mapCanvas->setVisible( true );
    m_mapCanvas->freeze( false );
    m_mapCanvas->refresh();
}
```

实现效果:

以加载 WMS 图层为例:





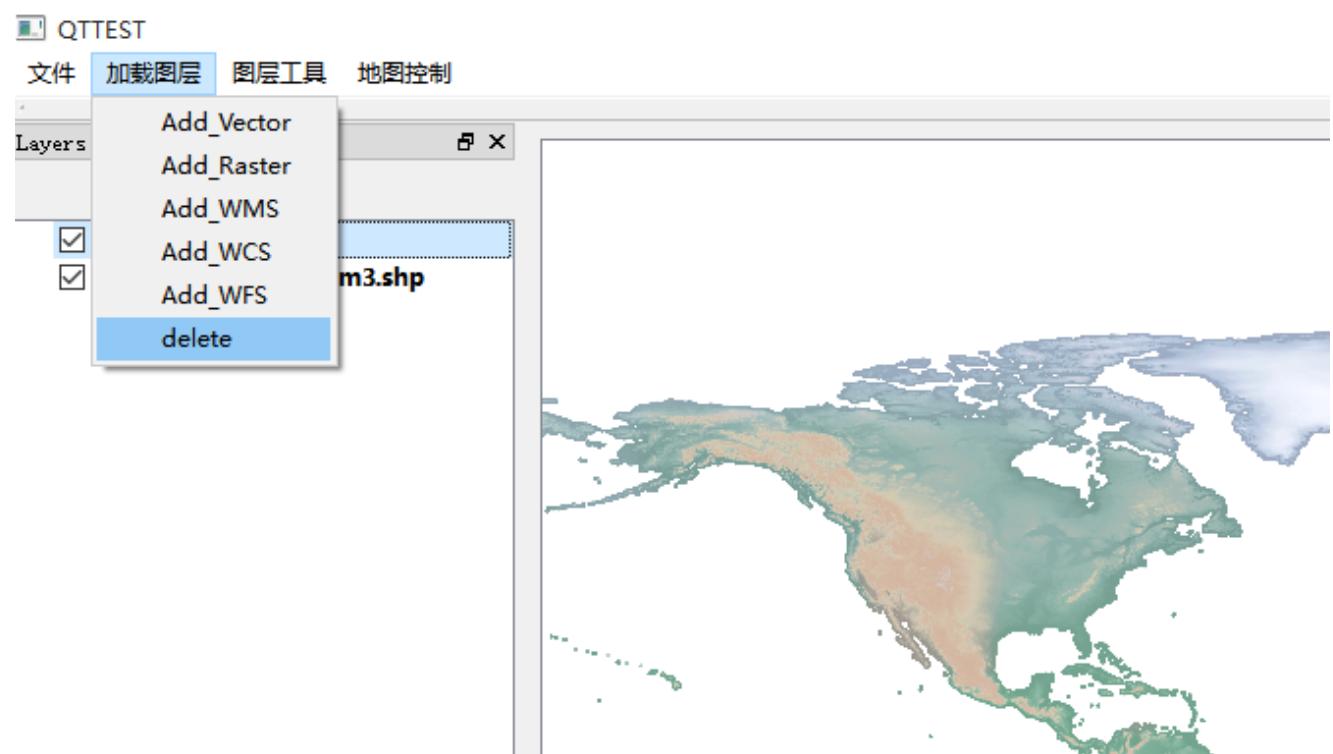
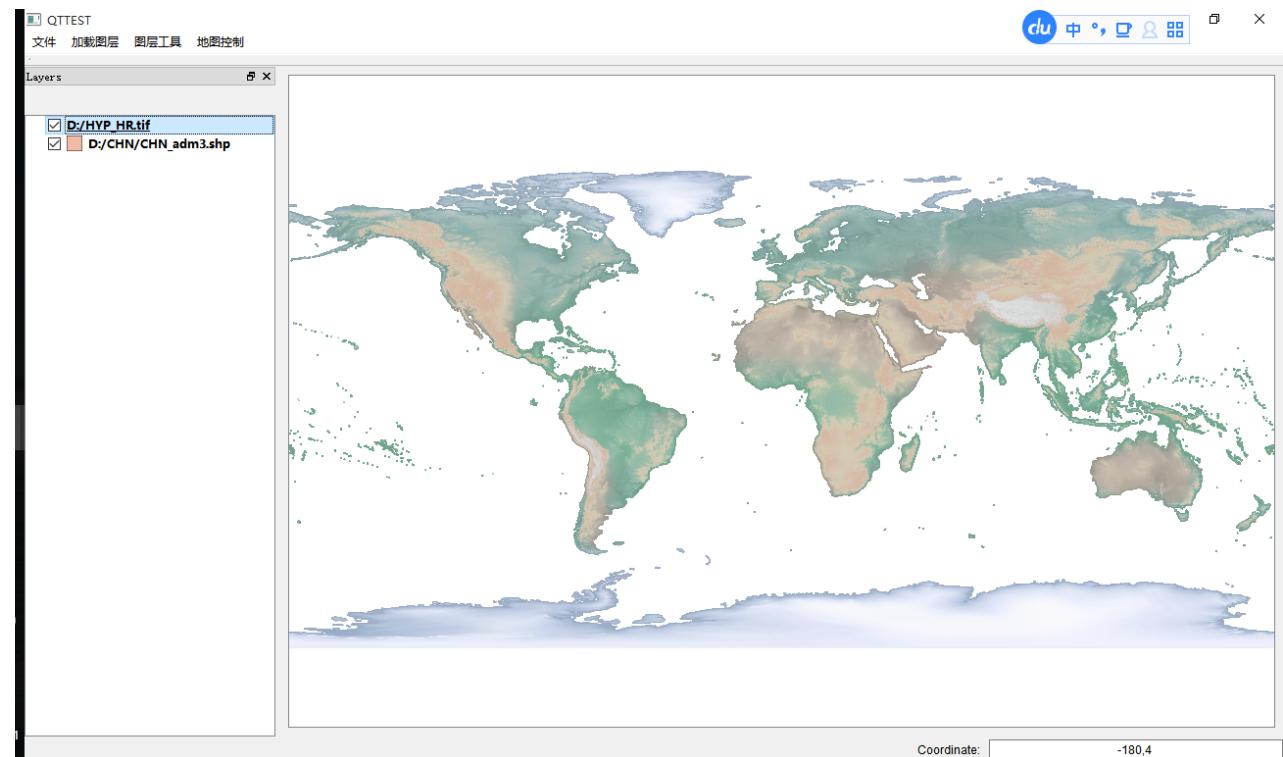
3.3.3 删除图层

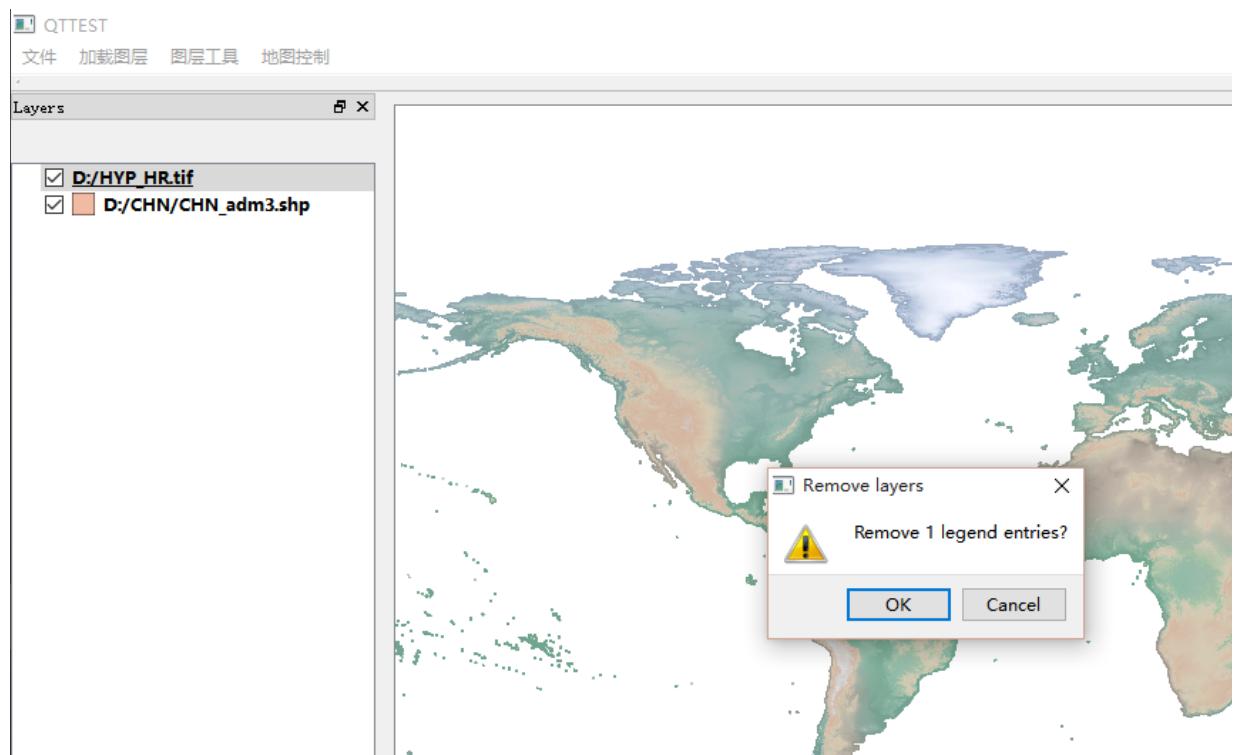
实现代码：

```
=====  
//删除图层  
void QTTEST::removeoneLayer()  
{  
    QLabel*label=new QLabel("删除！");  
    label->show();  
    if ( !m_layerTreeView ) {return;}  
    foreach( QgsMapLayer* layer, m_layerTreeView->selectedLayers() )  
    {  
        QgsVectorLayer* veclayer = qobject_cast<QgsVectorLayer*>( layer );  
        if ( veclayer && veclayer->isEditable() ) {return;}  
    }  
  
    QList<QgsLayerTreeNode*> selectedNodes = m_layerTreeView->selectedNodes( false );  
  
    // validate selection  
    if ( selectedNodes.isEmpty() )  
    {  
        QMessageBox::critical(  
            this,  
            tr( "Error" ),  
            tr( "No selection valid" ) );  
        return;  
    }  
    bool promptConfirmation = QSettings().value( "qgis/askToDeleteLayers", true ).toBool();  
    //display a warning  
    if ( promptConfirmation && QMessageBox::warning( this, tr( "Remove layers" ), tr( "Remove %n legend entries?", "number of legend i" ) ) == QMessageBox::Yes )  
    {  
        return;  
    }  
    foreach ( QgsLayerTreeNode* node, selectedNodes )  
    {  
        QgsLayerTreeGroup* parentGroup = qobject_cast<QgsLayerTreeGroup*>( node->parent() );  
        if ( parentGroup )  
        {  
            parentGroup->removeChildNode( node );  
        }  
    }  
    m_mapCanvas->refresh();  
}
```

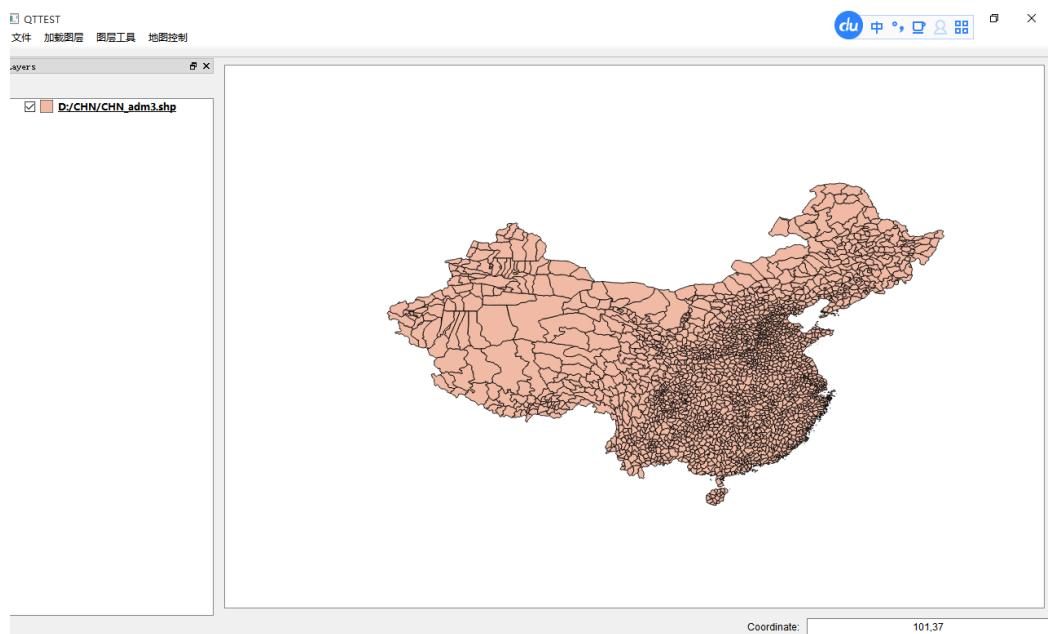
实现效果：

选中要删除的图层：





删除成功：



3.3.4 图层标注

实现代码：

```
/* **** */
/* 图层标注 */
void QTTEST::layerSymbolTest()
{
    // 获取当前选中的图层
    QgsVectorLayer* veclayer = qobject_cast<QgsVectorLayer*>( this->activeLayer() );
    if( !veclayer->isValid() ) { return; }

    if ( veclayer->geometryType() == QGis::Point )
    {
        // 创建 svgMarkerSymbolLayer
        QgsSvgMarkerSymbolLayerV2* svgMarker = new QgsSvgMarkerSymbolLayerV2( "money/money_b"

        QgsSymbolLayerV2List symList;
        symList.append( svgMarker );

        QgsMarkerSymbolV2* markSym = new QgsMarkerSymbolV2( symList );

        QgsSingleSymbolRendererV2* symRenderer = new QgsSingleSymbolRendererV2( markSym );

        svgMarker->setSize( 10 );
        veclayer->setRendererV2( symRenderer );
    }
}
```

在这里需要获得当前选中图层：

```
//获取当前选中的图层
QgsMapLayer* QTTEST::activeLayer()
{
    return m_layerTreeView ? m_layerTreeView->currentLayer() : 0;
}
```

配置标注属性：

```
/* **** */
//图层标注
void QTTEST::layerSymbolTest() [ ... ]
void QTTEST::testVecLayerLabel()
{
    QgsVectorLayer* layer = (QgsVectorLayer*)this->activeLayer();
    if (layer == NULL || layer->isValid() == false) { return; }

    // 首先是定义一个 QgsPalLayerSettings 变量，并启用他的属性设置
    QgsPalLayerSettings layerSettings;
    layerSettings.enabled = true;

    // 然后就可以开始根据API文档中的属性，进行自定义配置了
    layerSettings.fieldName = layer->pendingFields()[3].name(); // 设置Label图层
    layerSettings.centroidWhole = true; // 设置位置参考的中心点

    // Label 字体设置
    layerSettings.textColor = QColor(0, 0, 0); // 设置字体颜色
    layerSettings.textFont = QFont("Times", 12); // 设置字体和大小

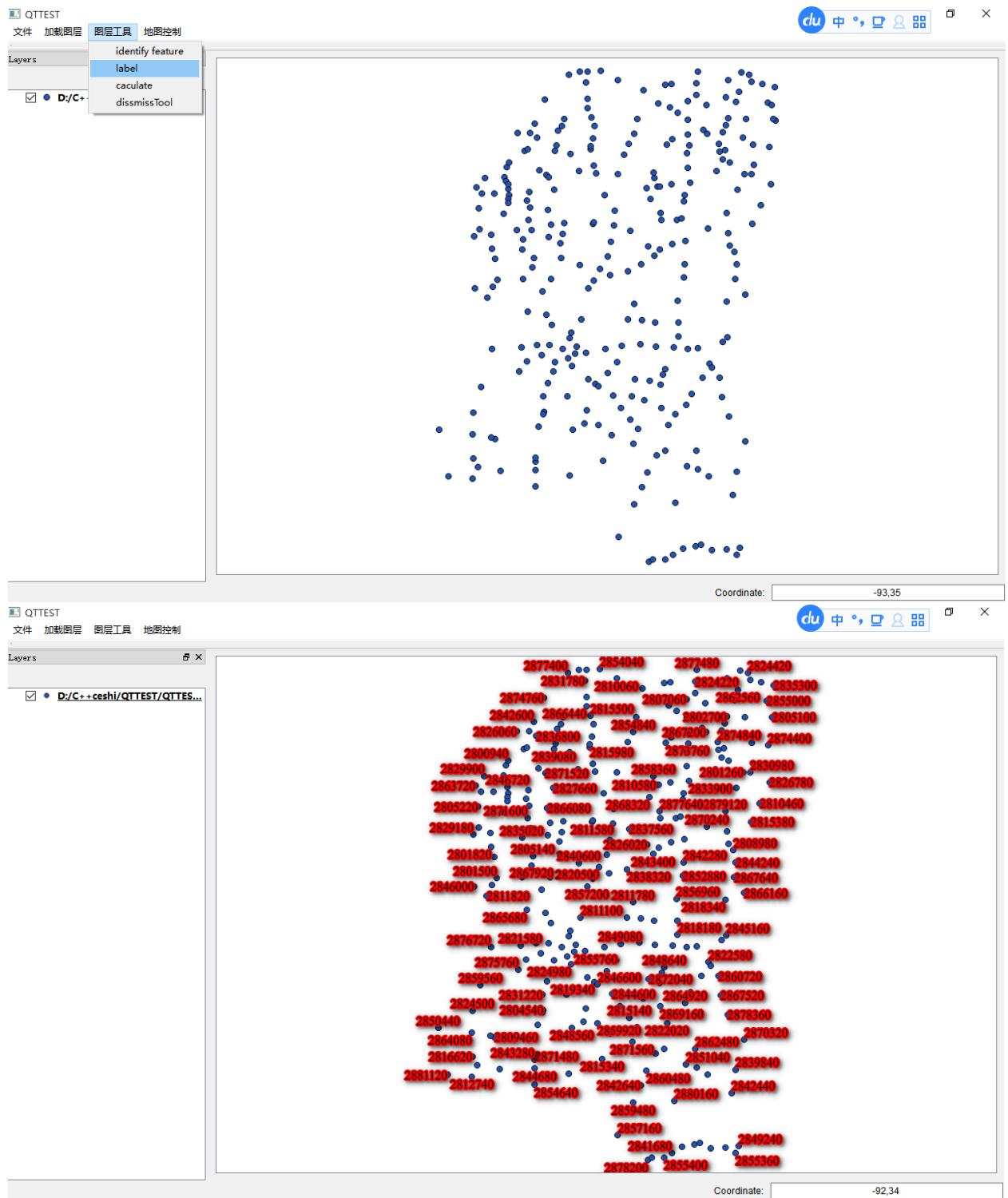
    // Label 轮廓buffer设置
    layerSettings.bufferDraw = true;
    layerSettings.bufferColor = QColor(255, 0, 0); // 轮廓buffer的颜色
    layerSettings.bufferSize = 1; // 轮廓buffer大小
    layerSettings.bufferTransp = 0.5; // 轮廓buffer的透明度

    // Label 阴影绘制
    layerSettings.shadowDraw = true;
    layerSettings.shadowOffsetAngle = 135; // 阴影的角度
    layerSettings.shadowOffsetDist = 1; // 阴影与Label的距离

    layerSettings.fieldName = layer->pendingFields()[3].name(); // 设置Label图层
    layerSettings.setDataDefinedProperty(layerSettings.Size, true, false, NULL, "size");
    layerSettings.setDataDefinedProperty(layerSettings.Color, true, false, NULL, "color");
    layerSettings.setDataDefinedProperty(layerSettings.Family, true, false, NULL, "font");

    layerSettings.writeToLayer(layer); // 将配置写入图层
    m_mapCanvas->refresh();
}
```

实现效果:



附加功能:

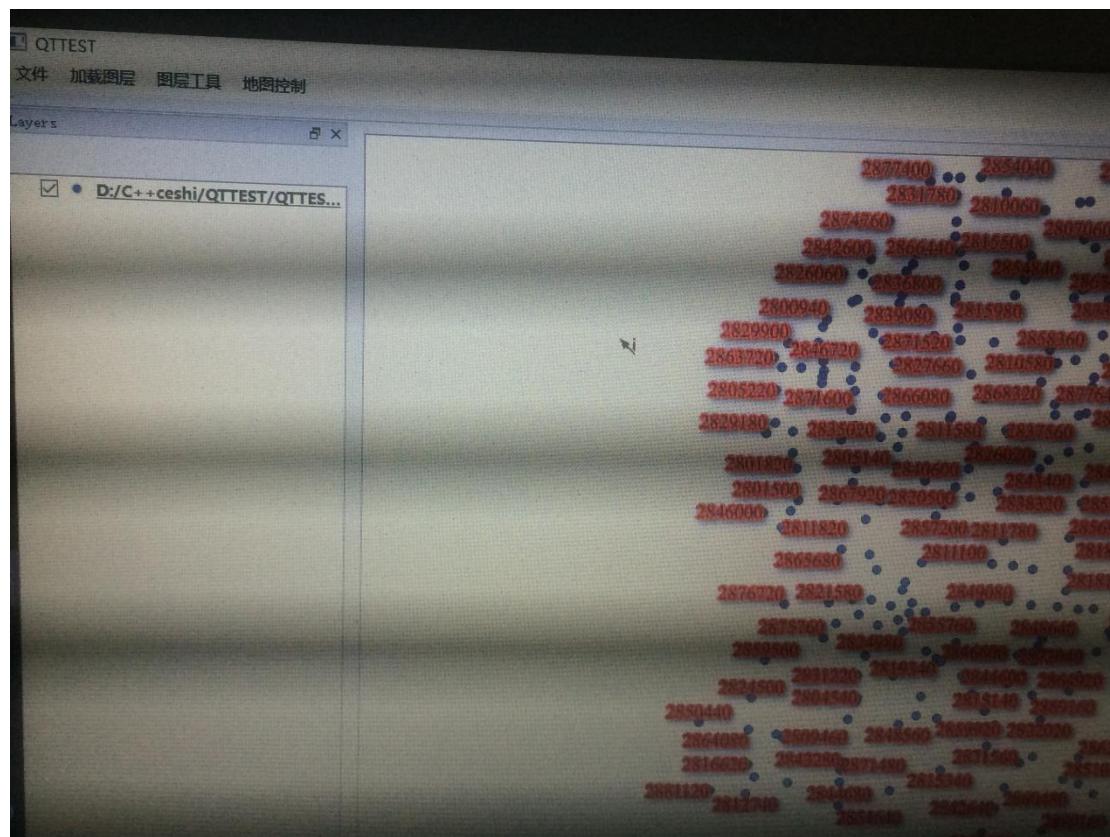
在图层工具的实现中，有一个问题就是，我选择了一个工具之后，如何取消这个工具的使用呢？于是就需要实现工具销毁功能，即将鼠标的属性从工具恢复为普

通的鼠标。

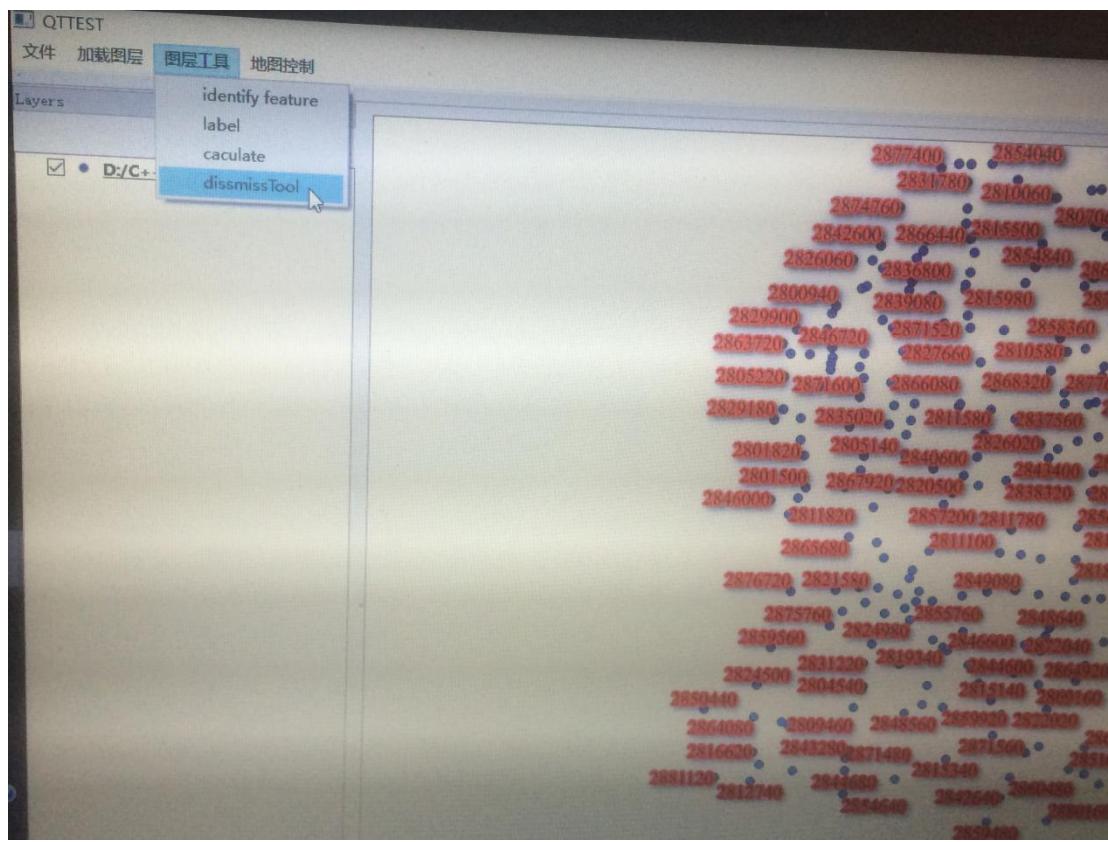
实现代码：

```
//*****
//取消地图工具
void QTTEST::dissstool()
{
    QgsMapTool *lastMapTool = m_mapCanvas->mapTool();
    m_mapCanvas->unsetMapTool( lastMapTool );
}
```

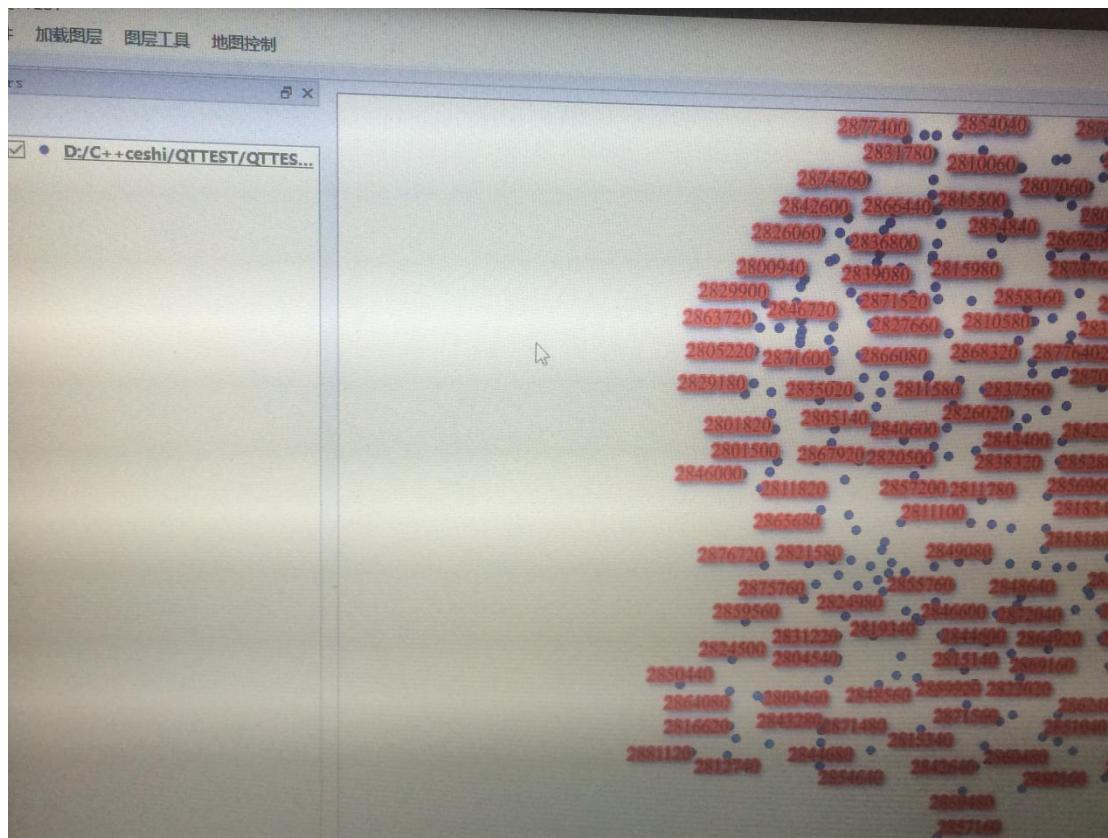
实现效果：



当前鼠标为图元识别状态，点击取消工具。



鼠标恢复为普通状态



！！！以上所有功能的调用，都是通过 connect 函数
响应按钮实现的：

```
// connections响应
connect( ui.actionAdd_Vector, SIGNAL( triggered() ), this, SLOT( addVectorLayers() ) );
connect( ui.actionAdd_Raster, SIGNAL( triggered() ), this, SLOT( addRasterLayers() ) );
connect( ui.action_2, SIGNAL( triggered() ), this, SLOT( openfile() ) );//打开项目
connect( ui.action_4, SIGNAL( triggered() ), this, SLOT( savefile() ) );//保存项目
connect( ui.action, SIGNAL( triggered() ), this, SLOT( newfile() ) );//新建项目
connect( ui.action_GIS, SIGNAL( triggered() ), this, SLOT( closefile() ) );//关闭项目
connect( ui.actionAdd_WMS, SIGNAL( triggered() ), this, SLOT( addWMSLayers() ) );//加载在线图层WMS
connect( ui.actionAdd_WFS, SIGNAL( triggered() ), this, SLOT( addWFSLayers() ) );//WFS
connect( ui.actionAdd_WCS, SIGNAL( triggered() ), this, SLOT( addWCSLayers() ) );//WCS
connect( ui.actionIdentify_feature, SIGNAL( triggered() ), this, SLOT( qIdentify() ) );//图元识别
connect( ui.actionDelete, SIGNAL( triggered() ), this, SLOT( removeoneLayer() ) );//删除图层
connect( ui.actionLabel, SIGNAL( triggered() ), this, SLOT( testVecLayerLabel() ) );//图层标注
connect( ui.actionDismissTool, SIGNAL( triggered() ), this, SLOT( disstool() ) );//图层标注
connect( ui.actionL, SIGNAL( triggered() ), this, SLOT( Large() ) );//放大
connect( ui.actionSmaller, SIGNAL( triggered() ), this, SLOT( Sma11() ) );//缩小
connect( ui.actionMove, SIGNAL( triggered() ), this, SLOT( Move() ) );//平移
connect( m_mapCanvas, SIGNAL( xyCoordinates( const QgsPoint& ) ), this, SLOT( showMouseCoordinate( cc
} )
```