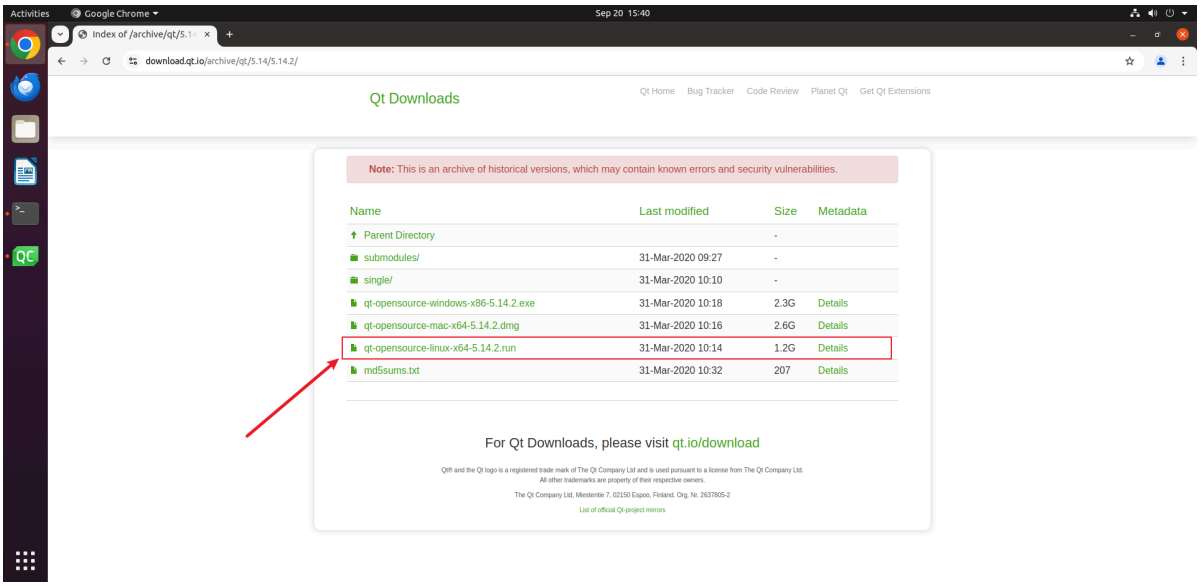
QT学习及记事本实现过程

一、获取Linux-Qt安装包

打开QT官网：Index of /archive/qt



选择5.14.2版本，复制qt-opensource-linux-x64-5.14.2.run链接下载，或者使用wegt下载

```
weget  https://download.qt.io/archive/qt/5.14/5.14.2/qt-opensource-linux-x64-5.14.2.run
```

二、安装qt

下载后在该安装包文件目录下执行命令：

```
sudo chmod +x qt-opensource-linux-x64-5.12.8.run
sudo ./qt-opensource-linux-x64-5.12.8.run
```

执行后注册qt账号，一直点Next，本次安装包我选择了全选，安装了所有包

三、配置环境变量

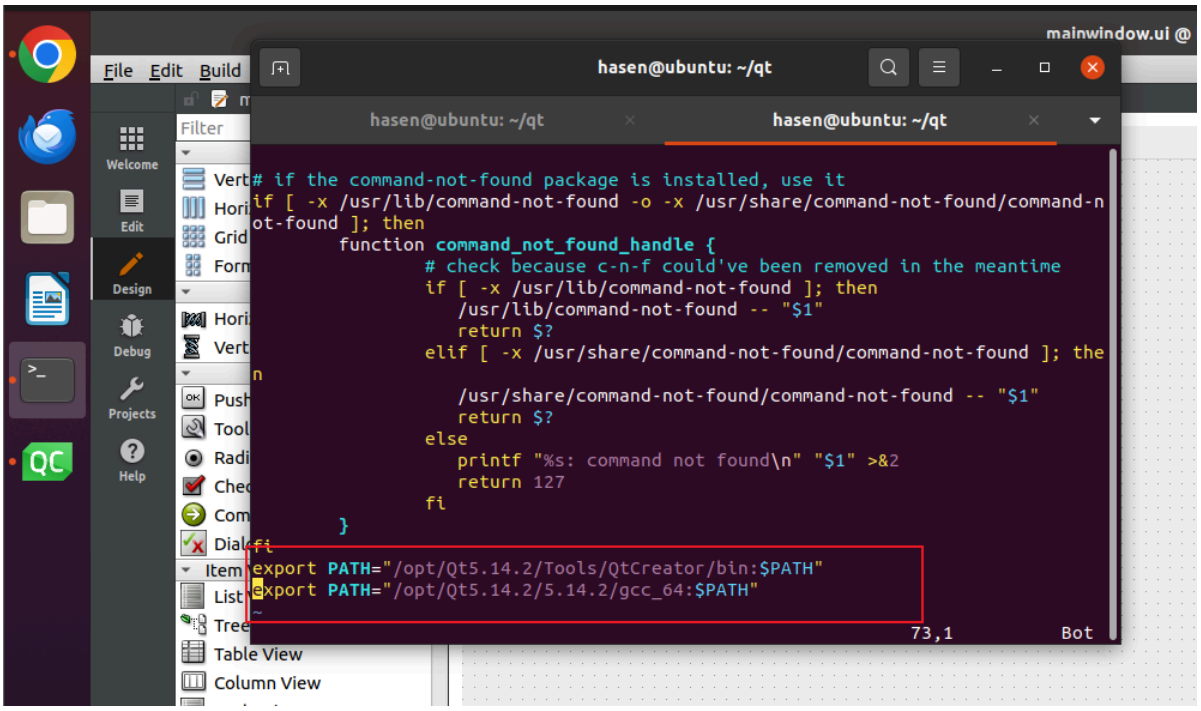
为了方便启动qtcreator，配置环境变量直接启动，在安装qt时使用sudo命令则安装在`/opt`路径下

使用命令如下：

```
sudo vim /etc/bash.bashrc
```

将如下配置写入

```
export PATH="/opt/Qt5.14.2/Tools/QtCreator/bin:$PATH"
export PATH="/opt/Qt5.14.2/5.14.2/gcc_64:$PATH"
```

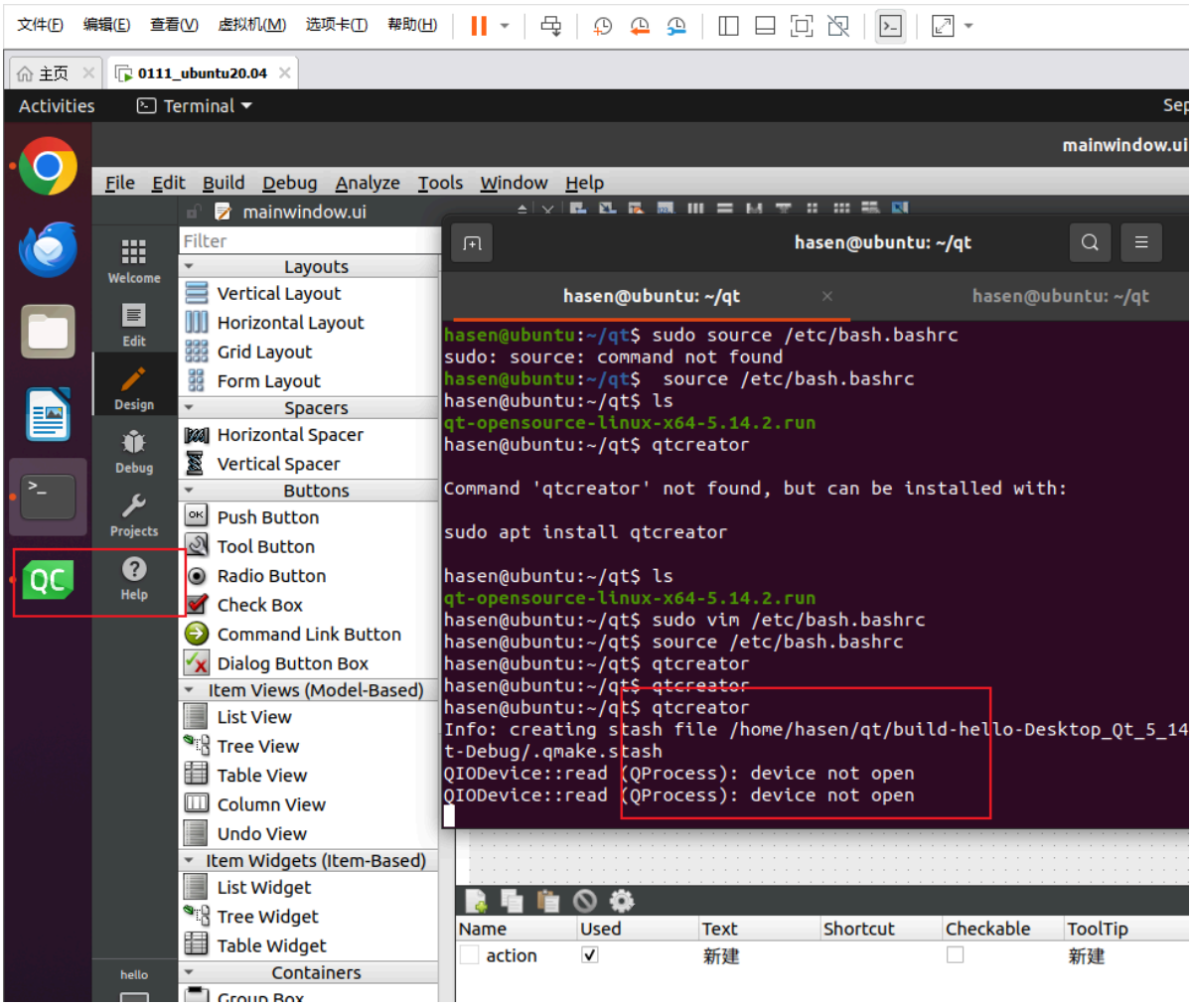


更新源：

```
source /etc/bash.bashrc
```

打开Qt如下图

```
qtcreator
```



安装GCC与G++

```
sudo apt-get install g++
sudo apt-get install build-essential
sudo apt-get install libgl1-mesa-dev
sudo apt-get install libglu1-mesa-dev freeglut3-dev
sudo apt-get install cmake
```

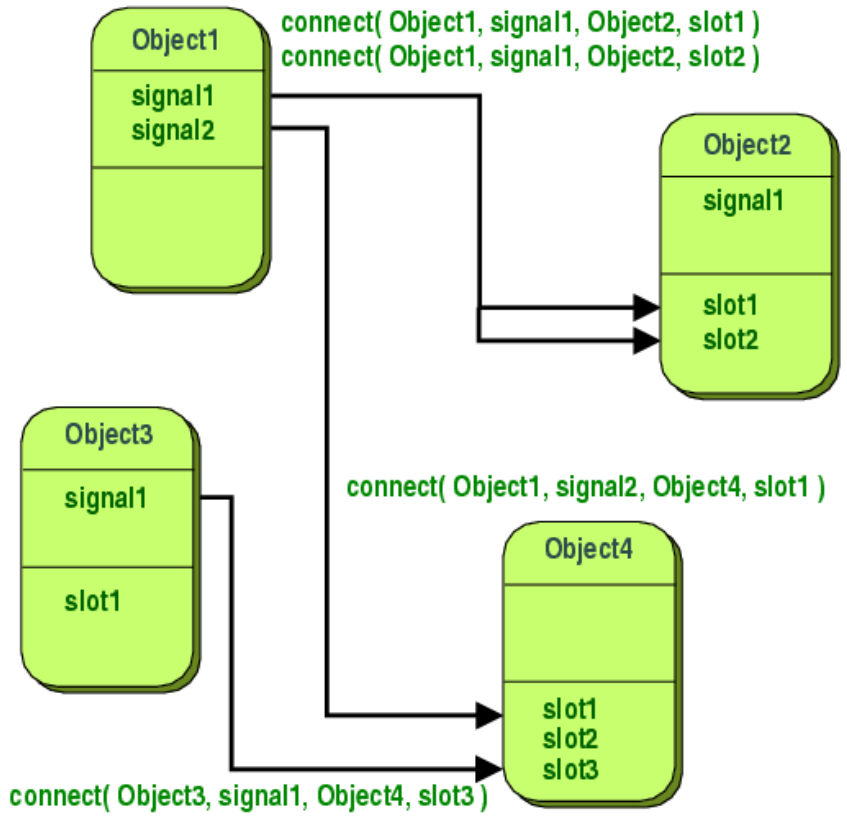
四、Qt基础学习

信号、槽、连接：

当特定事件发生时，会发出信号（Signal）。槽（Slot）是响应特定信号而调用的函数。信号和槽通过连接（Connect）来关联。这样的机制不知道也不关心哪些槽接收了信号，将程序正真的独立。可以将信号和槽的方法当成回调的一种替代，但是与回调相比，信号和槽略慢。

- 事件对象（`QEvent`）：封装了事件的信息，如事件的类型、触发源和相关数据。
- 事件循环：Qt 程序内部有一个事件循环，它不断检查和调度用户交互产生的事件，分配给相应的控件处理。
- 事件处理函数：控件（如 `QTextEdit`）可以通过重写父类的事件处理函数来响应特定事件，如 `wheelEvent()` 处理滚轮事件。

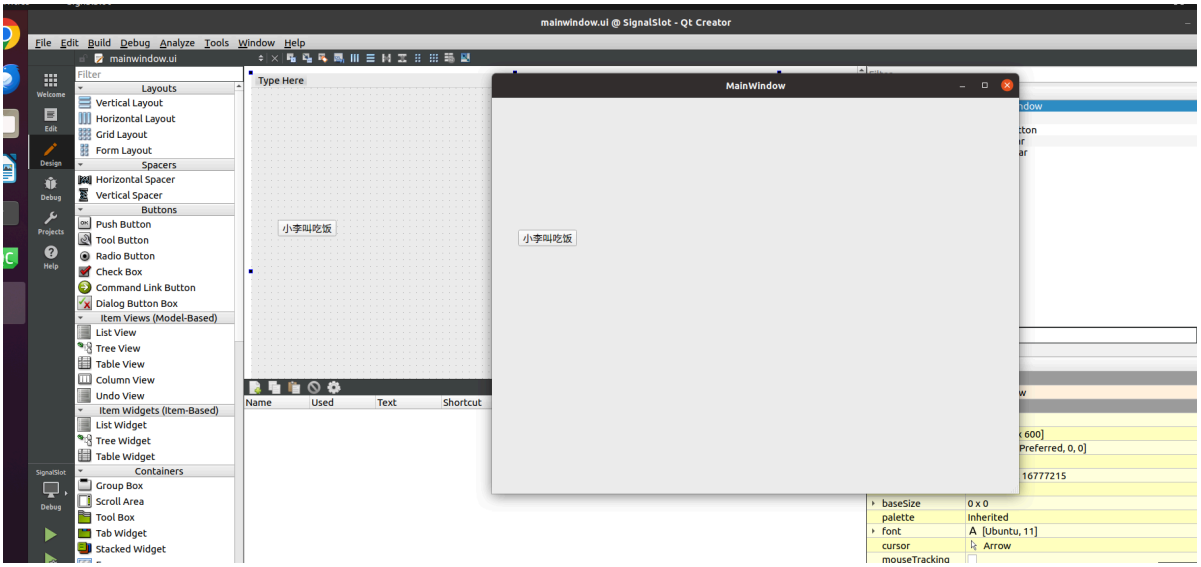
参考官方文档：https://doc.qt.io/qt-6/signalsandslots.html



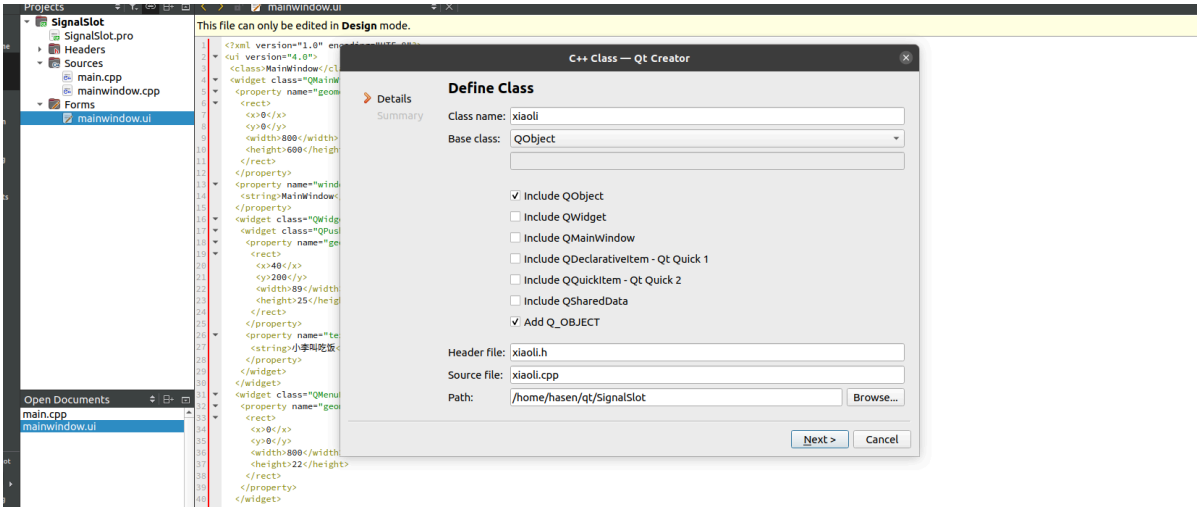
根据官方文档学习与阅读，编写示例加深对槽、连接、信号的理解。

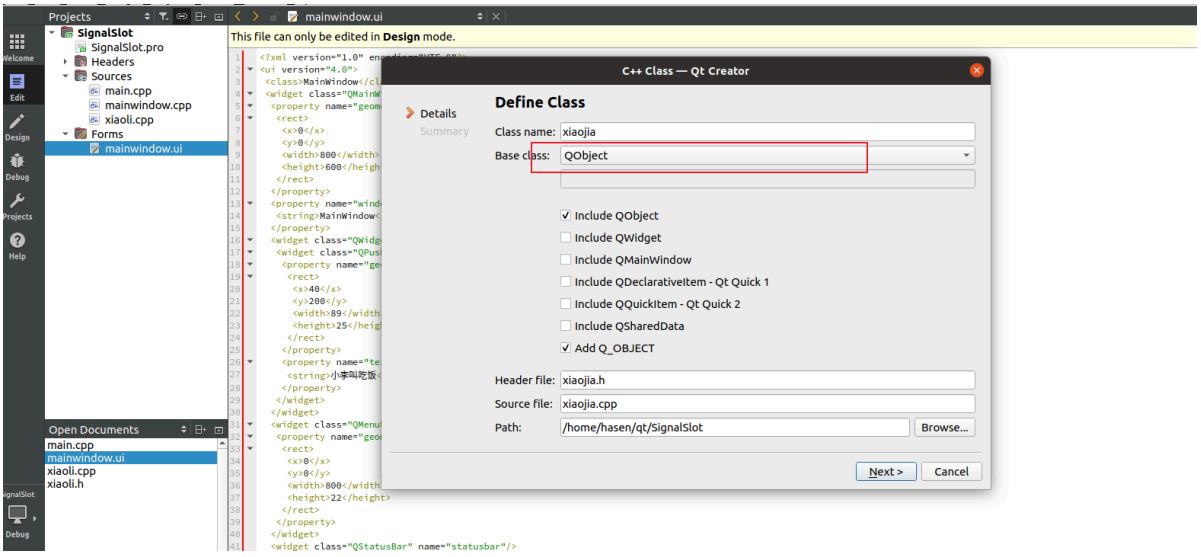
如下场景描述为：小李分别叫小贾和小红吃饭：

绘制按钮，并赋值“小李叫吃饭”



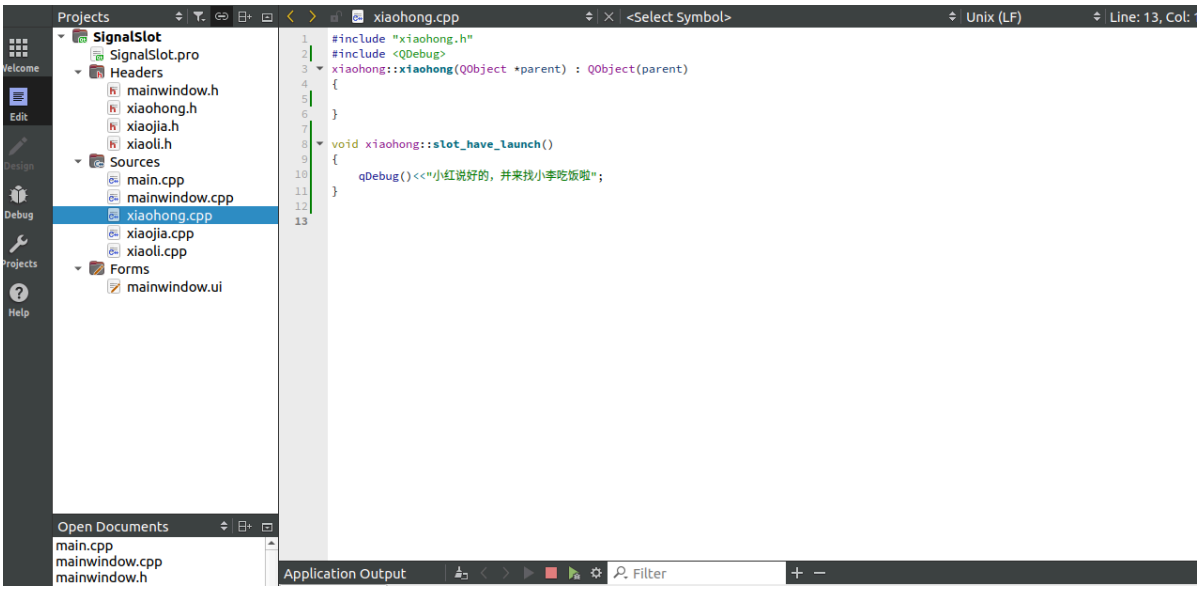
新建类，命名为xiaoli，并勾选继承QObject



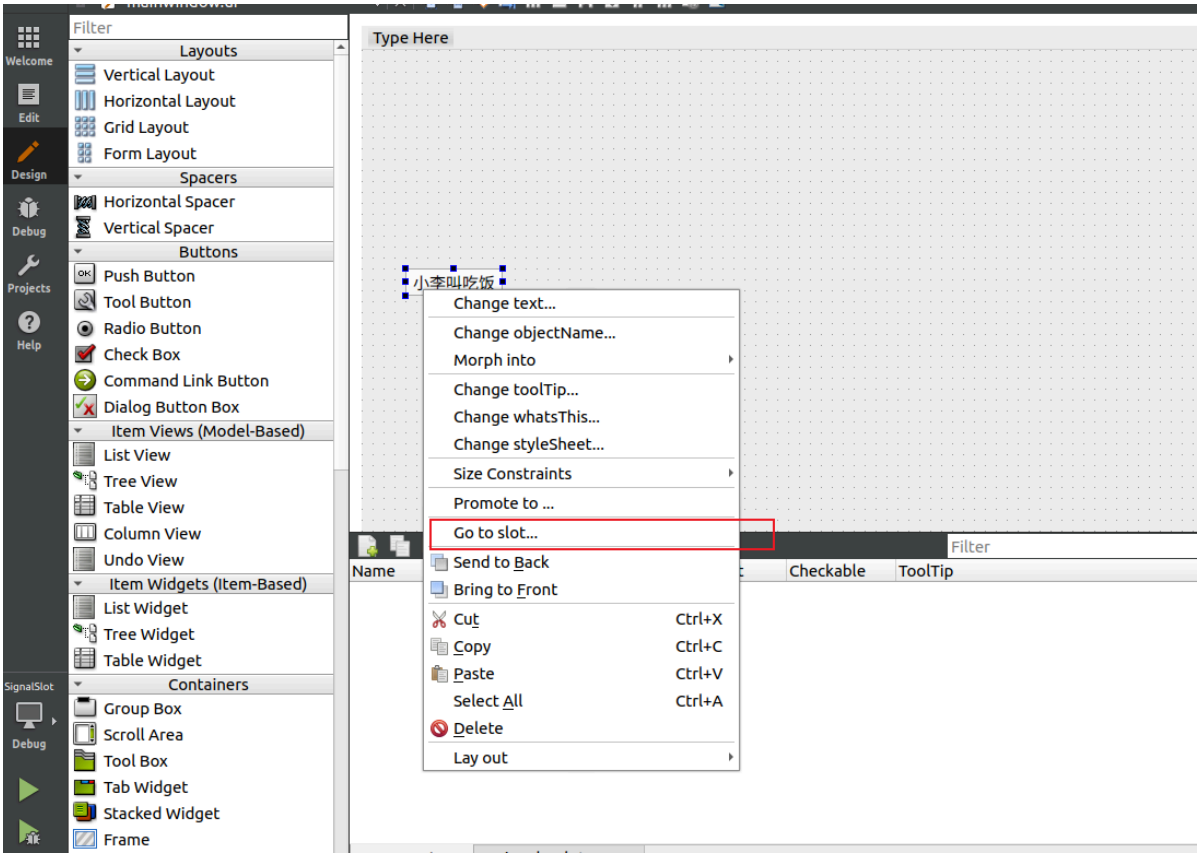


同理创建xiaojia，xiaohong两个类

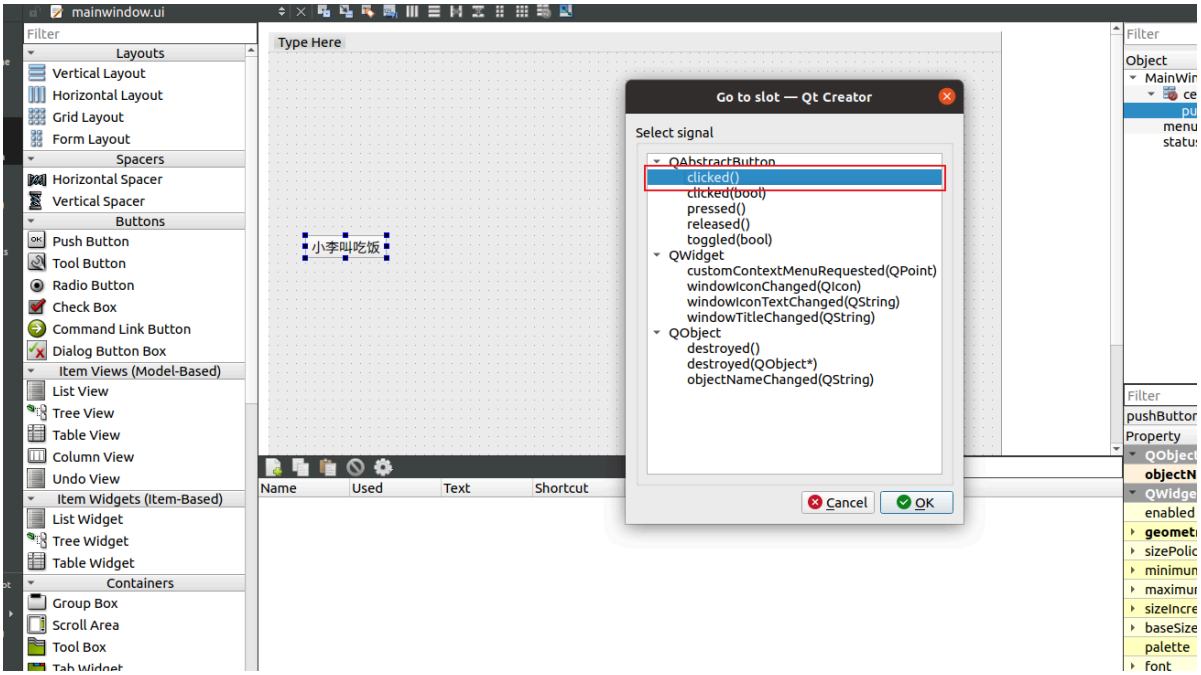
分别在.h文件中定义槽函数，右键，选择添加函数实现快捷方式，在槽函数写函数实现。



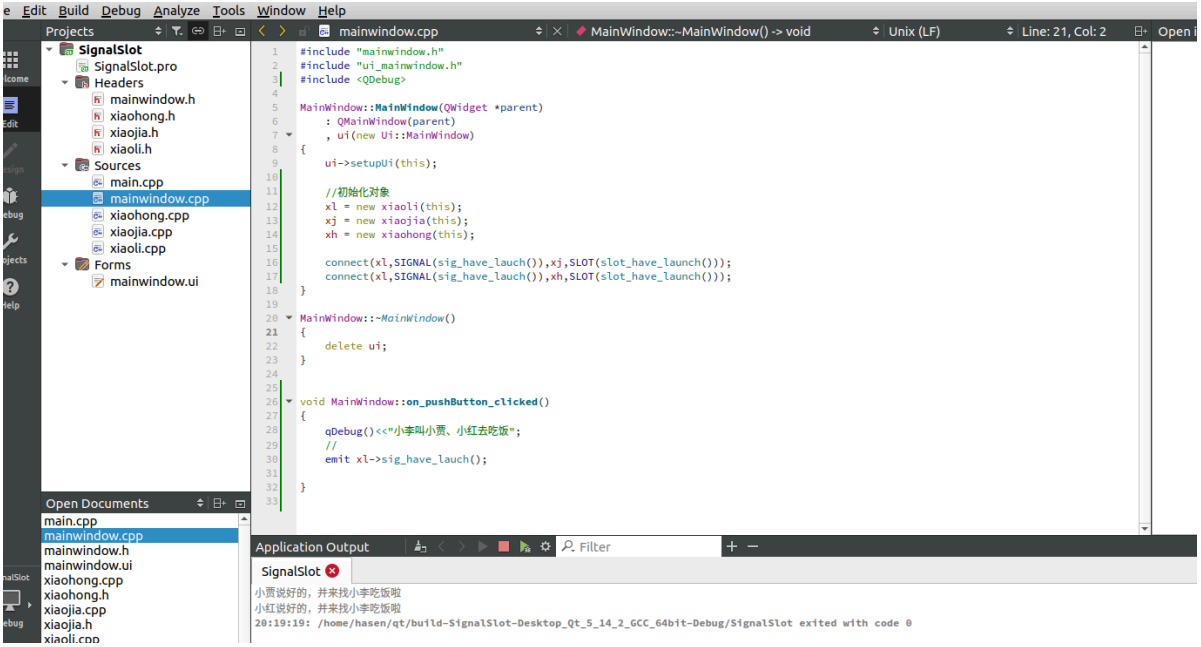
选择UI界面，选择按钮，右键到槽函数



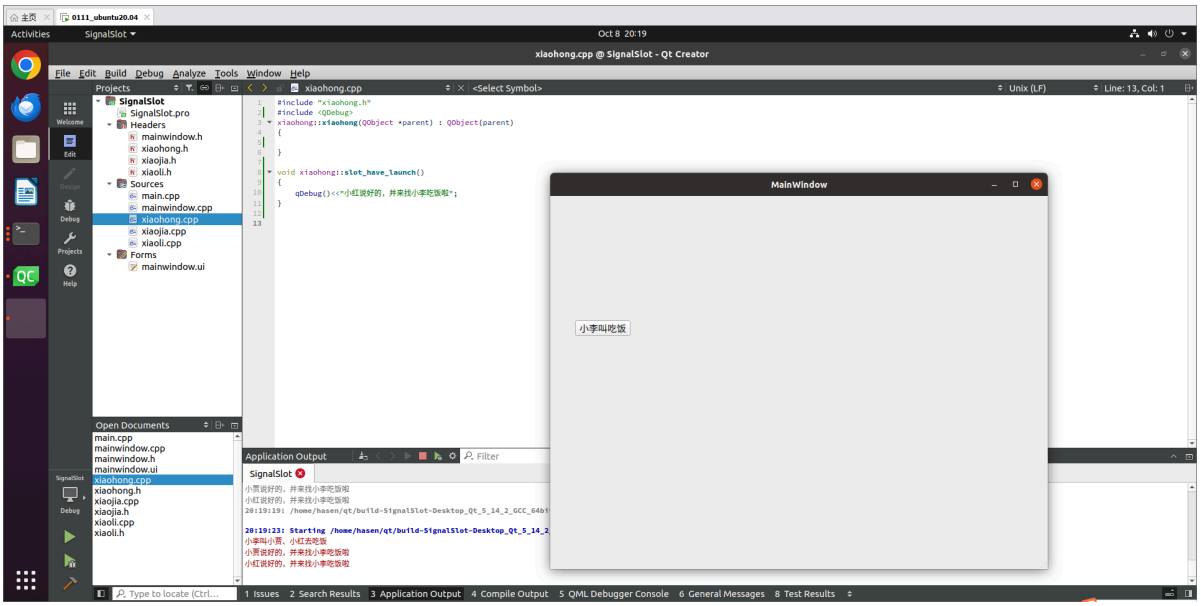
触发方式选择点击



在主函数写入实现及触发



运行效果如下



五、利用设计器绘制记事本UI

打开Qt设计器，由于无法输入中文在外部记事本或者其他可编辑程序中输入记事本需要的各种工具栏名称，复制到设计器中的工具栏中

主窗口包含：

1.菜单栏，一个主窗口只有一个菜单栏

2.工具栏，一个主窗口有多个工具栏

3.中心部件，一个主窗口只有一个中心部件

4.dock部件，停靠窗口，一个主窗口有多个dock部件

文件

新建
打开
保存
另存为
退出

编辑

撤销
剪切
复制
粘贴
删除

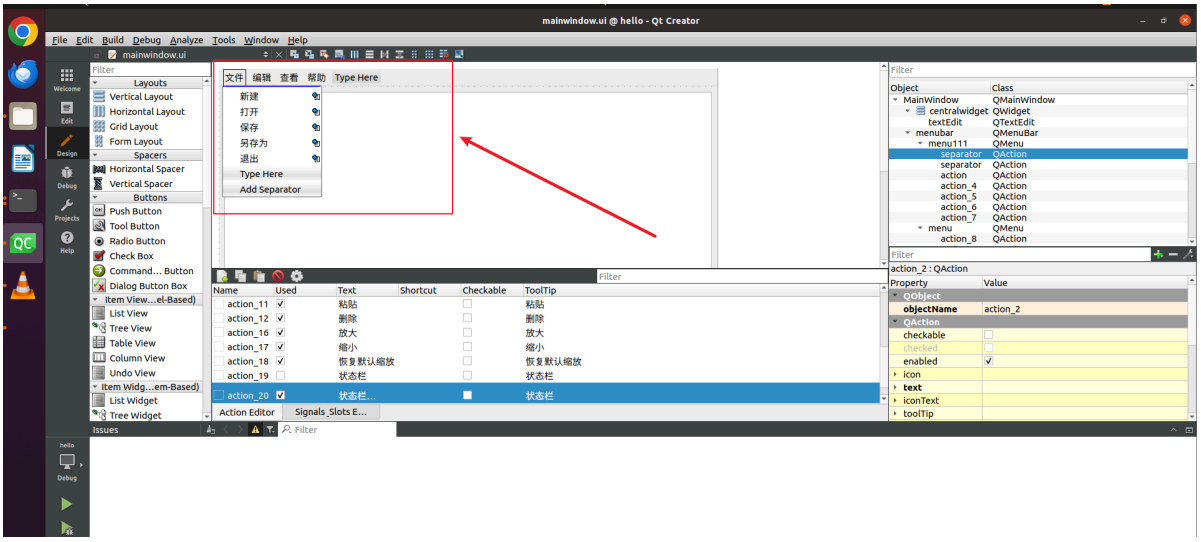
查看

缩放　放大　缩小　恢复默认缩放　状态栏

帮助

欢迎使用notepadplus

依次将上述的内容填充，如下



六、实现菜单栏各种功能

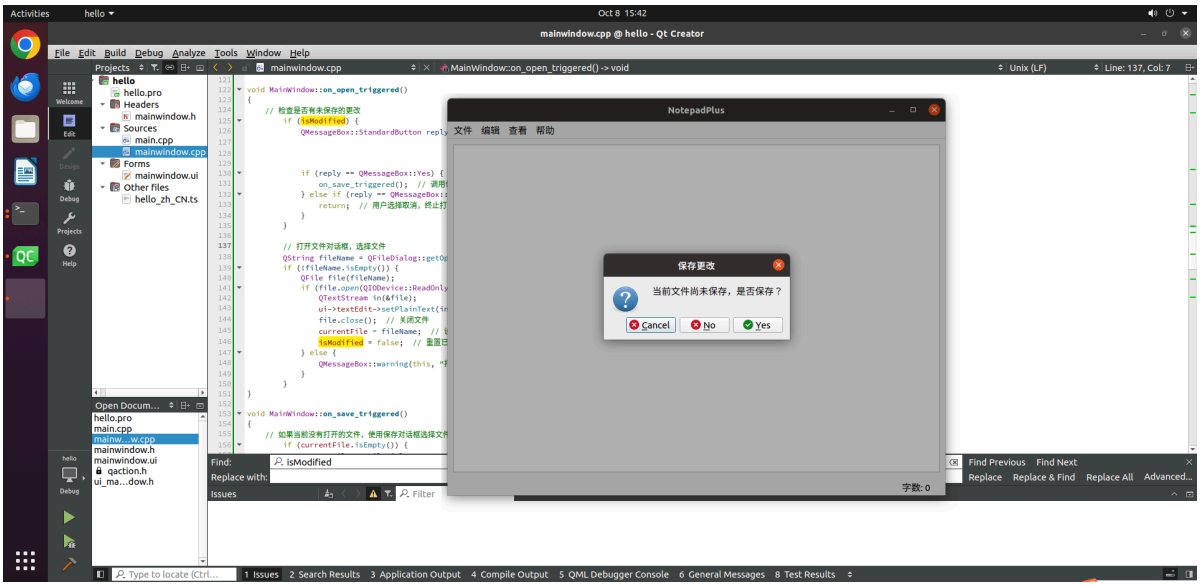
（1）新建

仿照windows记事本的逻辑，新建功能在点击后需要判断当前记事本是否存在未保存的文本，若未保存则提示保存并给出保存的选项，若无未保存的文本则清空当前文本内容

```
// 实现“新建”功能的槽函数
void MainWindow::on_newpage_triggered()
{
    // 检查当前内容是否已修改，并提示保存
    checkForUnsavedChanges();

    // 清空编辑区
    ui->textEdit->clear();

    // 重置已修改状态
    isModified = false;
}
```

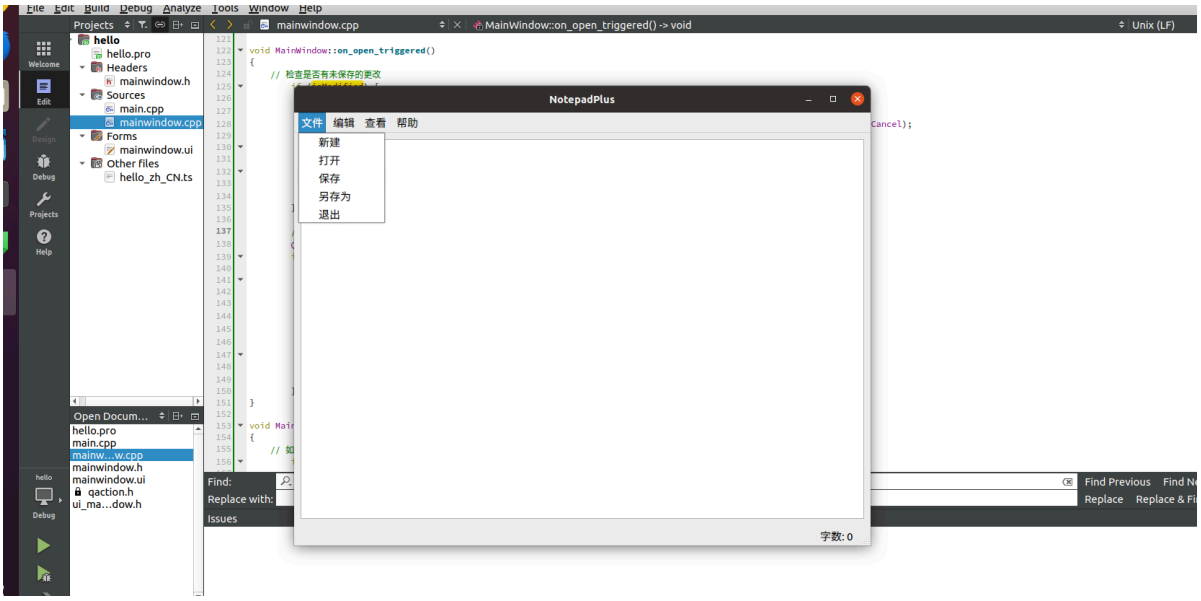
```
// 检查是否有未保存的更改
void MainWindow::checkForUnsavedChanges()
{
    if (isModified) {
        // 如果内容已修改，弹出提示框，询问用户是否保存
        QMessageBox::StandardButton reply = QMessageBox::question(this, "保存更改",
                                                                  "当前文件尚未保存，是否保存？",
                                                                  QMessageBox::Yes | QMessageBox::No | QMessageBox::Cancel);
        if (reply == QMessageBox::Yes) {
            // 如果用户选择保存，则调用保存函数（这里假设有 save 功能）
            saveFile();
        } else if (reply == QMessageBox::Cancel) {
            // 如果用户选择取消，则终止新建操作
            return;
        }
    }
}
```

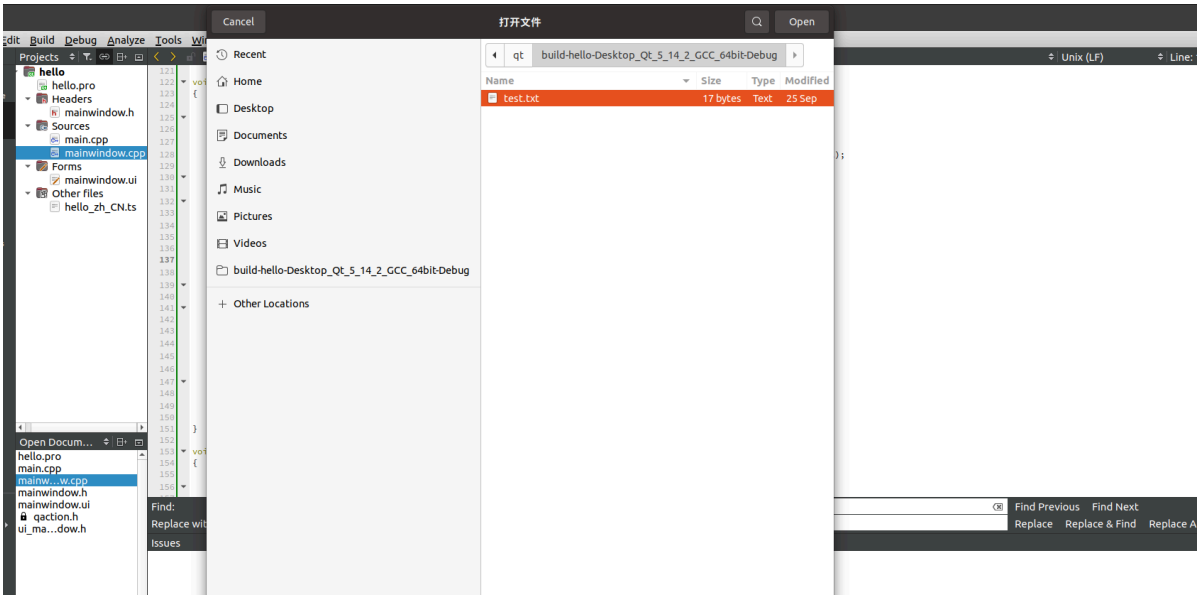
其中使用isModified标记是否修改，点击编译如图



（2）打开

同理当使用记事本打开新的文本文档时，也需要检查当前文本是否为空，若不为空检查是否保存，给出对应的选项进行保存或者打开的路径





（3）保存

点击保存弹出对应的路径，若打开的是已经存在的文本文档，修改后点击保存则保存到的是当前默认的文本文档。

```
void MainWindow::on_save_triggered()
{
    // 如果当前没有打开的文件，使用保存对话框选择文件名
    if (currentFile.isEmpty()) {
        currentFile = QFileDialog::getSaveFileName(this, "保存文件", "", "文本文件 (*.txt);;所有文件 (*)");
        if (currentFile.isEmpty()) {
            return;  // 用户取消保存
        }
    }

    // 尝试打开文件进行写入
    QFile file(currentFile);
    if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QTextStream out(&file);
        out << ui->textEdit->toPlainText();  // 将文本编辑区的内容写入文件
        file.close();  // 关闭文件
        isModified = false;  // 重置已修改状态
    } else {
        // 如果文件无法打开，显示错误提示
        QMessageBox::warning(this, "保存错误", "无法保存文件。");
    }
}
```

（4）另存为

检查当前用户选择的路径，若选择则进行保存写入，取消则直接返回，保存写入后保存当前路径


```
}

// 另存为
void MainWindow::on_saveas_triggered()
{
    // 获取文件路径
    QString fileName = QFileDialog::getSaveFileName(th

    if (fileName.isEmpty()) {
        return;
    }
    // 打开文件进行写操作
    QFile file(fileName);
    if (!file.open(QIODevice::WriteOnly | QIODevice::T
        QMessageBox::warning(this, tr("错误"), tr("无法
        return;
    }
    // 使用 QTextStream 写入文本内容
    QTextStream out(&file);
    out << ui->textEdit->toPlainText();
    file.close();
    currentFile = fileName;
    setWindowTitle(fileName);
}
void MainWindow::on_exit_triggered()
```

（5）退出

同理，退出与新建、保存、另存为等都有类似的逻辑，需要检查当前文件是否保存，根据isModified字段进行判断，使用QMessageBox给出提示

（6）撤销、删除、复制、粘贴、剪切

调用testEdit的功能分别实现，在mainWindow中声明

private slots:

```
void on_action_Y_triggered(); // 撤销
void on_action_X_triggered(); // 剪切
void on_action_C_triggered(); // 复制
void on_action_V_triggered(); // 粘贴
void on_action_D_triggered(); // 删除
```

（7）缩放功能

不使用鼠标滚轮依靠点击的方式实现缩放功能，这样比较麻烦，缩放需要多次点击。

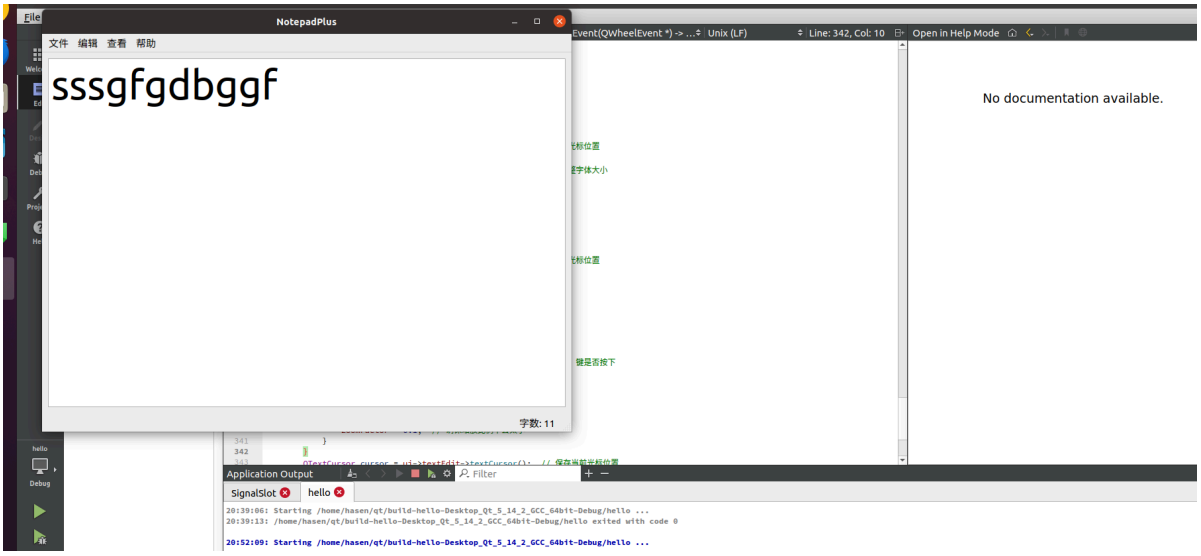
```
// 放大功能实现
void MainWindow::on_action_16_triggered()
{
    zoomFactor += 0.1;  // 增加缩放比例
    QTextCursor cursor = ui->textEdit->textCursor();  // 保存当前光标位置
    ui->textEdit->selectAll();  // 选中所有文本
    ui->textEdit->setFontPointSize(12 * zoomFactor);  // 按比例调整字体大小
    ui->textEdit->setTextCursor(cursor);  // 恢复光标位置
}

// 缩小功能实现
void MainWindow::on_action_17_triggered()
{
    zoomFactor -= 0.1;  // 减少缩放比例
    if (zoomFactor < 0.1) {
        zoomFactor = 0.1;  // 确保缩放比例不会太小
    }
    QTextCursor cursor = ui->textEdit->textCursor();  // 保存当前光标位置
    ui->textEdit->selectAll();  // 选中所有文本
    ui->textEdit->setFontPointSize(12 * zoomFactor);  // 按比例调整字体大小
    ui->textEdit->setTextCursor(cursor);  // 恢复光标位置
}

// 恢复默认缩放功能实现
void MainWindow::on_action_18_triggered()
```

```
{
    zoomFactor = 1.0;  // 重置缩放比例
    QTextCursor cursor = ui->textEdit->textCursor();  // 保存当前光标位置
    ui->textEdit->selectAll();  // 选中所有文本
    ui->textEdit->setFontPointSize(12);  // 恢复默认字体大小
    ui->textEdit->setTextCursor(cursor);  // 恢复光标位置
}
```

结合事件机制利用鼠标滚轮实现缩放，当用户使用鼠标滚轮时，系统会生成一个 `QWheelEvent` 事件对象，传递给当前鼠标指针所在的控件。Qt 的事件系统会根据鼠标指针位置，将事件传递给对应的控件，并调用该控件的 `wheelEvent` 处理函数。

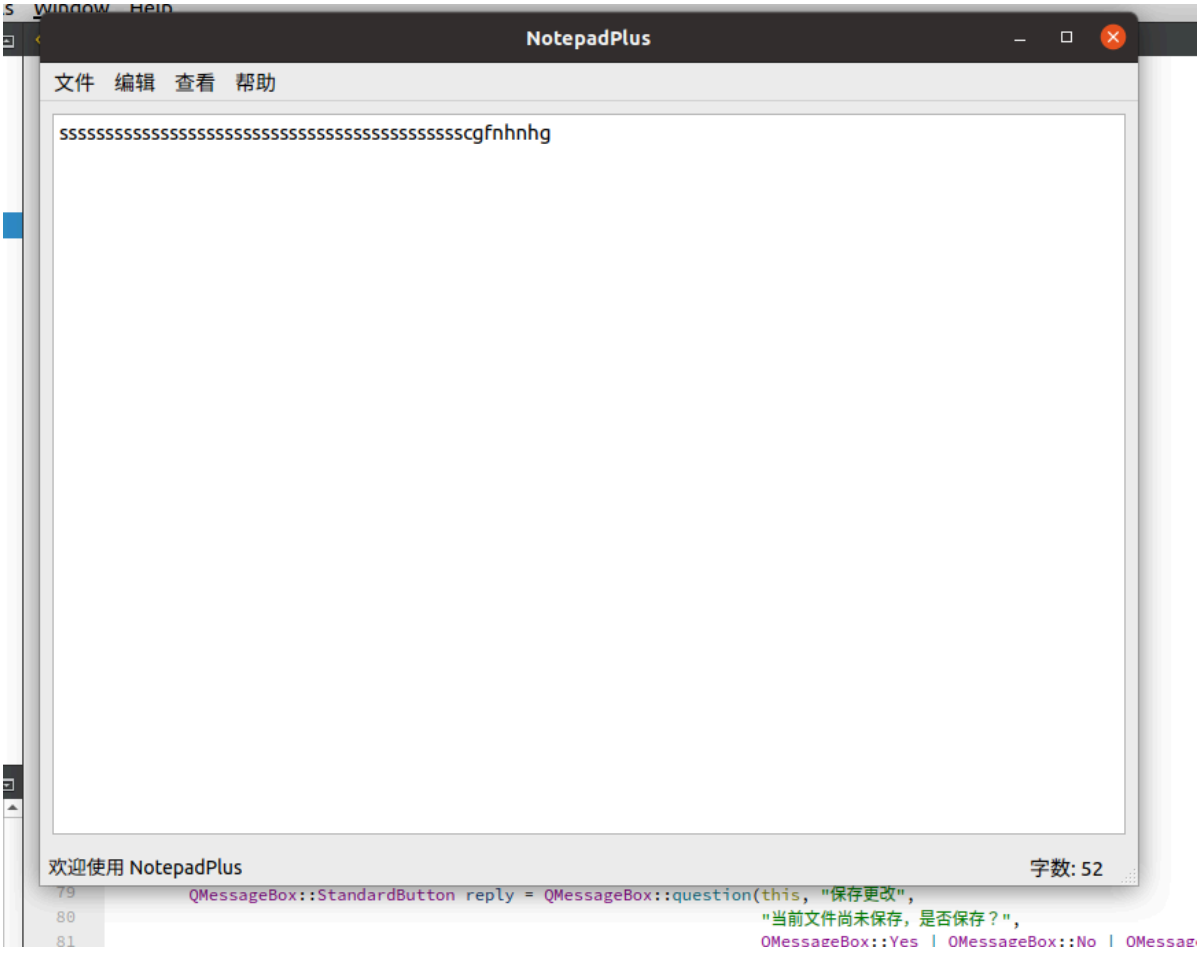
```
// 放大功能实现
void MainWindow::on_action_16_triggered()
{
    zoomFactor += 0.1;  // 增加缩放比例
    QTextCursor cursor = ui->textEdit->textCursor();  // 保存当前光标位置
    ui->textEdit->selectAll();  // 选中所有文本
    ui->textEdit->setFontPointSize(12 * zoomFactor);  // 按比例调整字体大小
    ui->textEdit->setTextCursor(cursor);  // 恢复光标位置
}

// 缩小功能实现
void MainWindow::on_action_17_triggered()
{
    zoomFactor -= 0.1;  // 减少缩放比例
    if (zoomFactor < 0.1) {
        zoomFactor = 0.1;  // 确保缩放比例不会太小
    }
    QTextCursor cursor = ui->textEdit->textCursor();  // 保存当前光标位置
    ui->textEdit->selectAll();  // 选中所有文本
    ui->textEdit->setFontPointSize(12 * zoomFactor);  // 按比例调整字体大小
    ui->textEdit->setTextCursor(cursor);  // 恢复光标位置
}

// 恢复默认缩放功能实现
void MainWindow::on_action_18_triggered()
{
    zoomFactor = 1.0;  // 重置缩放比例
    QTextCursor cursor = ui->textEdit->textCursor();  // 保存当前光标位置
    ui->textEdit->selectAll();  // 选中所有文本
    ui->textEdit->setFontPointSize(12);  // 恢复默认字体大小
    ui->textEdit->setTextCursor(cursor);  // 恢复光标位置
}

// 处理滚轮缩放事件
void MainWindow::wheelEvent(QWheelEvent *event)
{
    if (event->modifiers() & Qt::ControlModifier) {  // 检查 Ctrl 键是否按下
        if (event->angleDelta().y() > 0) {
            zoomFactor += 0.1;  // 向上滚动放大
        } else {
            zoomFactor -= 0.1;  // 向下滚动缩小
            if (zoomFactor < 0.1) {
```

```
                zoomFactor = 0.1;  // 确保缩放比例不会太小
            }
        }
        QTextCursor cursor = ui->textEdit->textCursor();  // 保存当前光标位置
        ui->textEdit->selectAll();
        ui->textEdit->setFontPointSize(12 * zoomFactor);
        ui->textEdit->setTextCursor(cursor);

        event->accept();  // 接受事件，防止其被其他部件处理
    } else {
        QMainWindow::wheelEvent(event);  // 调用父类的默认处理
    }
}
```



(8) 状态栏字数统计

```
// 更新字数统计的槽函数
void MainWindow::updateWordCount()
{
    // 获取 QTextEdit 中的文本
    QString text = ui->textEdit->toPlainText();

    // 计算字数
    int wordCount = text.length();

    // 更新字数显示
    wordCountLabel->setText("字数: " + QString::number(wordCount));
}
```



实现字数的实时统计

(9)帮助、版本等显示

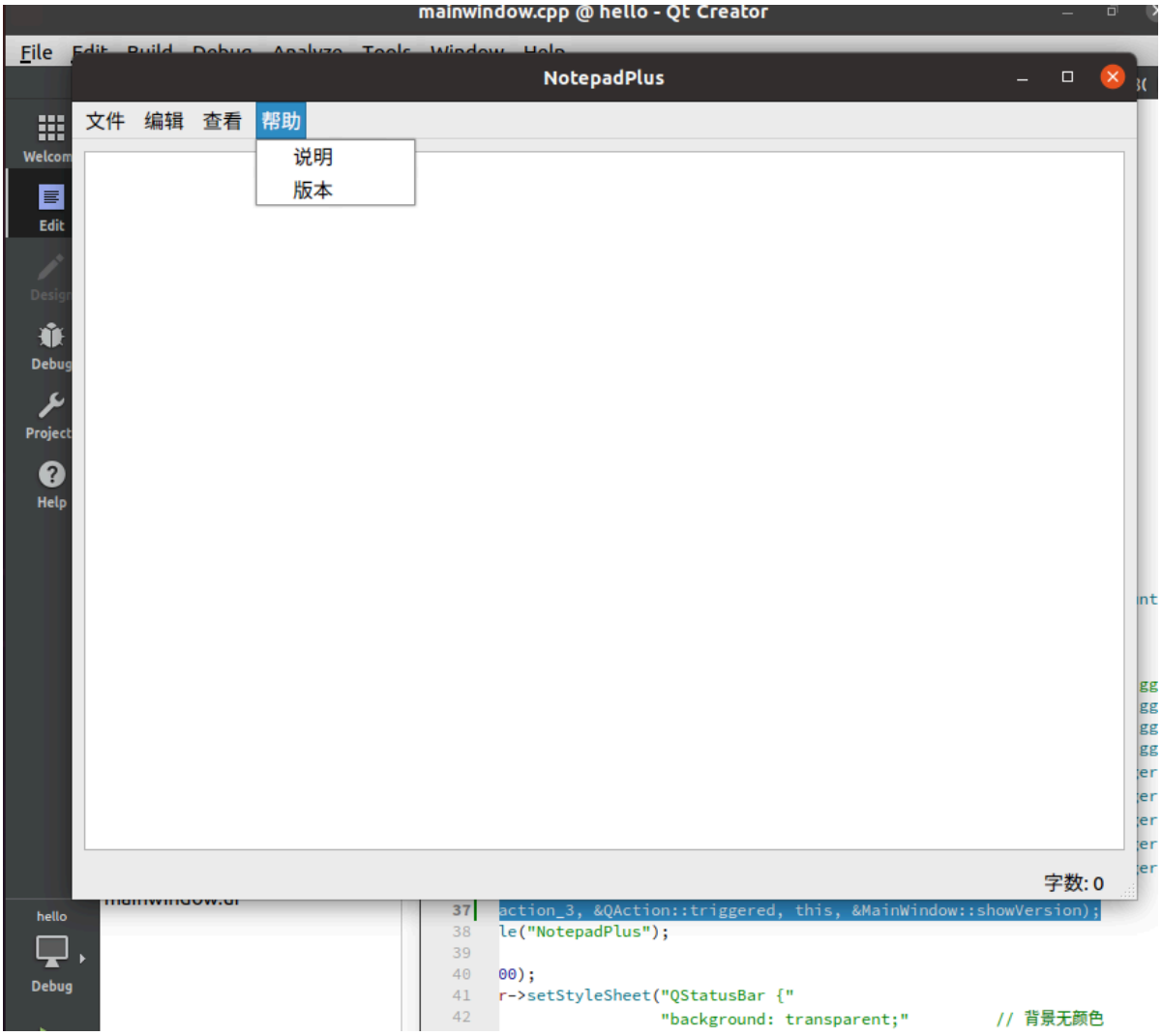
利用基本的信号与槽实现

连接:

```
connect(ui->action_2, &QAction::triggered, this, &MainWindow::showHelp);
connect(ui->action_3, &QAction::triggered, this, &MainWindow::showVersion);
```

函数实现:

```
void MainWindow::showHelp()
{
    QMessageBox::information(this, "说明", "缩放功能请使用Ctrl+鼠标滚轮。");
}

void MainWindow::showVersion()
{
    QMessageBox::information(this, "版本", "版本 1.0.0\n开发者: 小李");
}
```



七、打包

在Linux中可以打包为deb或者appimage，本项目打包为appimage。

工具及环境安装

安装QT5开发工具qmake

```
sudo apt update
sudo apt install qt5-qmake qtbase5-dev
qmake -version
sudo apt-get install qttools5-dev-tools

make//编译
qmake hello//
```

下载 `.AppImage` 文件,赋予执行权限

```
wget
https://github.com/probonopd/linuxdeployqt/releases/download/continuous/linuxdeployqt-continuous-x86_64.AppImage
chmod a+x linuxdeployqt-continuous-x86_64.AppImage
```

按照上述流程执行后，会生成`default.desktop`文件，`.desktop`文件是 Linux 桌面环境中用于描述应用程序的文件。修改名为`default.desktop`的文件。内容如下：

```
[Desktop Entry]
Version=1.0
Name=notepade+
Exec=hello
Icon=hello
Type=Application
Categories=Utility;
```

- `Name`是应用程序显示的名称。
- `Exec`是可执行文件的名称。
- `Icon`是应用程序图标的名称（可以为 PNG 或 SVG 格式文件，具体路径或名称与提供的图标文件匹配）。

重新运行`linuxdeployqt`

在项目根目录下，确保有：

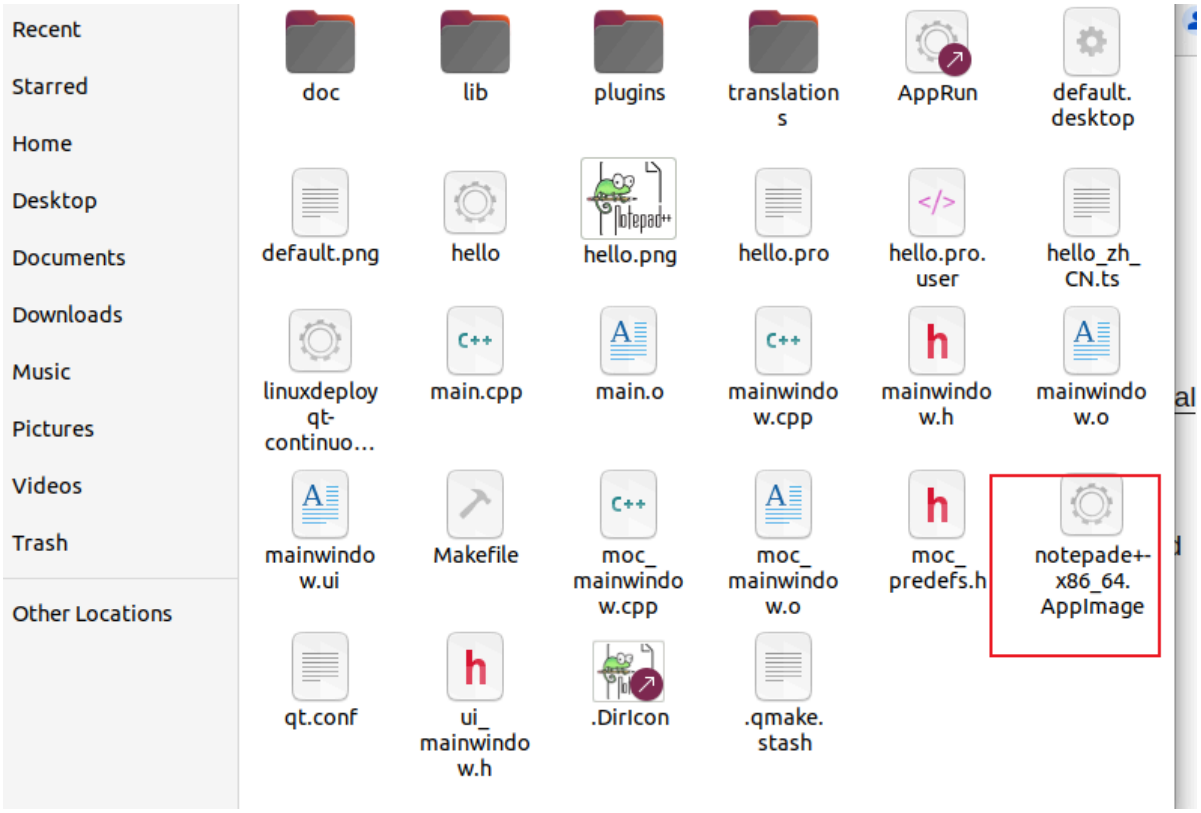
- 编译好的可执行文件`hello`
- `.desktop`文件
- 图标文件

然后重新运行`linuxdeployqt`：

```
./linuxdeployqt-continuous-x86_64.AppImage hello -appimage
```

运行`.AppImage`文件：
通过终端运行以下命令或者直接双击该文件以启动应用程序

```
./notepad+.AppImage
```



双击上图的文件即可运行记事本，如下图