我们最终到达了QT入门的最后一篇文章。

本节是对QT的另一个关键概念的学习，即信号槽机制。一个对象发出信号（各种事件），另一个对象响应信号（槽函数）并处理。

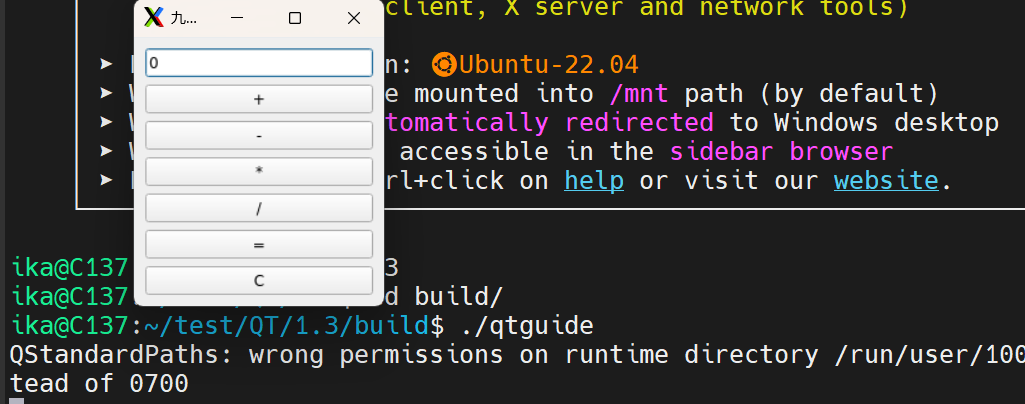
|  |
| --- |
| *class* MainWindow : *public* QMainWindow{  *private:*      QLineEdit \*m\_lineEdit;      //枚举类型拥有全局的命名空间，可以直接使用，当然仅限public  *enum* Operator{          None,          Plus,          Minus,          Multiply,          Divide,      };  *double* m\_lastValue;      Operator m\_lastOperator = None;  *public:*      explicit MainWindow(QWidget \**parent* = nullptr) : QMainWindow(*parent*){          setWindowTitle("九九乘法表");          setGeometry(400, 400, 200, 200);          setCentralWidget(new QWidget);          QGridLayout \*layout = new QGridLayout();          //QT中的get函数不需要使用get，set函数需要使用set          centralWidget()->setLayout(layout);          m\_lineEdit = new QLineEdit;          m\_lineEdit->setText("0");          m\_lineEdit->setValidator(new QDoubleValidator); //设置验证器，只能输入数字          layout->addWidget(m\_lineEdit);          QList<QPair<Operator, QString>> operators = {              {Plus, "+"},              {Minus, "-"},              {Multiply, "\*"},              {Divide, "/"},          };          //结构化绑定，直接将pair中的两个值绑定到op和name中          for(*auto* const &[op, name] : operators) {              QPushButton \*button = new QPushButton;              button->setText(name);              layout->addWidget(button);              connect(button, &QPushButton::clicked, this, [this, op](){                  this->pressOperator(op);              });          }          QPushButton \*equalButton = new QPushButton;          equalButton->setText("=");          layout->addWidget(equalButton);          connect(equalButton, &QPushButton::clicked, this, &MainWindow::pressEqual);          QPushButton \*zeroButton = new QPushButton;          zeroButton->setText("C");          //layout->addWidget(zeroButton, 1, 0);          layout->addWidget(zeroButton);          connect(equalButton, &QPushButton::clicked, this, [this]{              this->m\_lineEdit->setText("0");              m\_lastOperator = None;          });      }  *void* setLineEditText(QString *text*){  *int* pos;          if (m\_lineEdit->validator()->validate(*text*, pos) != QValidator::Invalid){              m\_lineEdit->setText(*text*);          }      }  *void* pressOperator(Operator *op*){  *double* value = m\_lineEdit->text().toDouble();          m\_lastValue = value;          m\_lastOperator = *op*;          m\_lineEdit->setText("0");      }  *void* pressEqual(){  *double* value1 = m\_lastValue;  *double* value2 = m\_lineEdit->text().toDouble();  *double* value;          switch(m\_lastOperator){              case Plus:                  value = value1 + value2;                  break;              case Minus:                  value = value1 - value2;                  break;              case Multiply:                  value = value1 \* value2;                  break;              case Divide:                  value = value1 / value2;                  break;              case None:                  value = value2;                  break;          }          m\_lineEdit->setText(QString::number(value));      }  }; |

connect函数的第一个参数是发送信号的对象，第二个参数是信号，第三个参数是接收信号的对象，第四个参数是槽函数。其中，第四个参数可以是一个函数，也可以是一个lambda表达式，也可以是一个函数对象

在信号槽机制中，无论是哪一个对象被销毁，connect函数都会自动断开连接

这样就避免了使用野指针导致内存泄漏的问题。比如，槽函数是一个函数对象，在首次收到信号后，delete掉这个对象。那么下次信号触发必然导致内存泄漏。事实上不是这样的，QT不会再触发槽函数，不会发生任何泄漏。

槽函数可以是lambda表达式，[this, op]表示捕获this指针和op变量。强推使用按值捕获，并注明捕获那些对象。因为，如果按引用捕获的话，在回调函数可能会导致悬空引用的问题，所以最好明确指定捕获方式。



本次程序，一个不带键盘的计算器。

本篇结束。