**程序设计开发实践报告**

**题 目 中国象棋**

**序 号 02**

**学 号 20232241091**

**姓 名 蔡佳兴**

**任课教师 范晓娅，于鲲鹏**

**成 绩**

大连理工大学软件学院

2024年7月

**目录**

[评分细则及标准 2](#_Toc143947979)

[1 技术调研报告 3](#_Toc143947980)

[1.2 第二周学习总结 3](#_Toc143947985)

[1.2.1 内容简介 3](#_Toc143947986)

[1.2.2 难点和解决办法 6](#_Toc143947987)

[1.2.3 学习案例](#_Toc143947988) 7

**评分细则及标准**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **考察项目** | **总分** | **评分细则** | **分数** | **得分** |
| **技术调研（第一周）** | 20分 | 调研报告 | 10分 |  |
| 平时表现 | 10分 |
|  |  |  |  |  |
| **技术调研（第二周）** | 20分 | 调研报告 | 10分 |  |
| 平时表现 | 10分 |
|  | | | | |
| **项目开发评分（总分60分）** | | | |  |
| 问题规模 | 10分 | 创新超额完成指定任务，工作量饱满 | 8-10分 |  |
| 基本完成指定任务，工作量一般 | 6-8分 |
| 指定任务未完成，工作量不足 | 0-5分 |
| 技术难度 | 10分 | 模型设计合理优化，采用合适的方法实现 | 8-10分 |  |
| 模型设计基本正确，采用较合适的方法实现 | 6-8分 |
| 模型设计存在问题，采用方法不合理 | 0-5分 |
| 实现程度 | 20分 | 系统实现完整，界面友好，测试全面无误 | 8-10分 |  |
| 系统实现完整，界面友好，存在少许错误 | 6-8分 |
| 系统实现不完整，界面不友好，存在错误 | 0-5分 |
| 报告质量 | 10分 | 报告完整、格式统一、结构清晰、图表正确 | 8-10分 |  |
| 报告较为规范、结构较为清晰、图表基本正确 | 6-8分 |
| 内容不完整不规范、结构不清晰、图表有错误 | 0-5分 |
| 项目汇报 | 10分 | 阐述清晰准确，回答问题准确到位 | 8-10分 |  |
| 阐述基本准确，回答问题基本准确 | 6-8分 |
| 阐述不够清晰完整，回答问题不准确 | 0-5分 |
|  |  |  |  |  |
| 最终得分 | 100分 |  | | |

# 1 技术调研报告

## 1.2 第二周学习总结

### 1.2.1 内容简介

事件处理方法：

缺省实现

void QWidget::keyPressEvent(QKeyEvent \*event) {

event->ignore(); }

重写缺省实现

void MyLineEdit::keyPressEvent(QKeyEvent \*event) {

qDebug() << event->text();

if(event->text() != " "){

QLineEdit::keyPressEvent(event); } }

事件过滤器：一个QObject管理另一个QObject实例的事件。

重新实现QApplication::notify() 方法需要2个步骤：

1 继承QApplication

2 重写notify方法

事件的传递：

Qt中,事件的派发是从 QApplication::notify() 开始。 • 因为QAppliction也是继承自QObject, 所以先检查QAppliation对象, 如 果有事件过滤器安装在qApp上, 先调用这些事件过滤器。 • 接下来QApplication::notify() 会过滤或合并一些事件(比如失效widget 的鼠标事件会被过滤掉, 而同一区域重复的绘图事件会被合并)。 • 事件被送到receiver,如果有事件过滤器, 先调用这些事件过滤器。 • 执行receiver ::event() 处理。 • 执行receiver 的特定事件处理函数，如 receiver::keyPressEvent。 • 对于鼠标, 滚轮, 按键等事件如果在整个事件的派发过程结束后还没有被 处理, 那么这个事件将会向上转发给它的父widget, 直到最顶层窗口。

QTimer自定义定时器**：**

启动定时器代码：

// 创建一个新的定时器

QTimer \*timer = new QTimer(this);

// 关联定时器的溢出信号到我们的槽函数上

connect(timer,SIGNAL(timeout()), this,SLOT(timerUpdate()));

// 设置溢出时间为1秒，并启动定时器

timer->start(1000);

定时器槽函数代码：

QDateTime datetime = QDateTime::currentDateTime();

QString str = datetime.toString("yyyy-MM-dd hh:mm:ss.zzz ddd");

ui->label->adjustSize(); ui->label->setText(str);

随机数：

qsrand()：设定随机数种子

qrand()：取得随机数

// 为随机数设置初值

qsrand(QTime(0,0,0).secsTo(QTime::currentTime()));

int rand = qrand()%100;

信号和槽的连接：

bool QObject::connect ( const QObject \* sender, const char \* signal, const QObject \* receiver, const char \* member )

信号和槽的断开：

有三种情况必须使用disconnect()函数：

1. 断开与某个对象相关联的任何对象

disconnect( myObject, 0, 0, 0 )或者myObject->disconnect()

1. 断开与某个特定信号的任何关联

disconnect( myObject, 0, 0, 0 )或者myObject->disconnect()

disconnect( myObject, SIGNAL(mySignal()), 0, 0 ) 或者

myObject->disconnect( SIGNAL(mySignal()) )

（3） 断开两个对象之间的关联

disconnect( myObject, 0, myReceiver, 0 )或者

myObject->disconnect( myReceiver )

使用属性：

直接获取

•QString text = label->text();

•label->setText("Hello World!");

通过元信息和属性系统

•QString text = object->property("text").toString();

•object->setProperty("text", "Hello World");

在运行时发现属性

•int QMetaObject::propertyCount();

•QMetaProperty QMetaObject::property(i);

•QMetaProperty::name/isConstant/isDesignable/read/write/...

对象树与拥有权：

•QObject \*parent = new QObject();

•QObject \*child1 = new QObject(parent);

•QObject \*child2 = new QObject(parent);

•QObject \*child1\_1 = new QObject(child1);

•QObject \*child1\_2 = new QObject(child1);

•delete parent;

**线程**(QThread)

QThread代表在应用程序中可以独立控制的线程，他与进程中的其他线程共享数 据，但是独立运行。

QThread::run() ：子类重写Run函数，实现线程逻辑。 QThread::start() ：启动线程，它内部默认调用run函数。

QThread提供的信号（ Q\_SIGNALS ）：

void started(); 线程启动后发射 void finished(); 线程结束后发射 void terminated(); 线程终止后发射

QThread提供的休眠函数：

static void sleep(unsigned long)： 按秒为单位休眠

static void msleep(unsigned long)：按毫秒为单位休眠

static void usleep(unsigned long)： 按微秒为单位休眠

同步函数：

bool wait()：让子线程和主线程同步

线程内的事件机制：

int exec() ：开启线程内的事件循环

exit() 或 quit():退出事件循环

**互斥量（**QMutex**）**

QMutex类提供的是线程之间的访问顺序化, 它的目的是保护一个对象、数据结构 或者代码段，所以同一时间只有一个线程可以访问它。

•QMutex::QMutex ( bool recursive = FALSE )

– 构造一个新的互斥量。这个互斥量是在没有锁定的状态下创建的。如果 recursive为真，就构造一个递归互斥量，如果recursive为假（默认值）， 就构造一个普通互斥量。对于一个递归互斥量，一个线程可以锁定一个互 斥量多次并且只有在相同数量的unlock调用之后，它才会被解锁。

•QMutex::locked () ：

– 锁定一个互斥量

•QMutex::unlock () ：

– 解锁这个互斥量

•QMutex::tryLock ()：

–试图锁定互斥量。如果锁被得到，这个函数返回真。如果另一个进程已经 锁定了这个互斥量，这个函数返回假，而不是一直等到这个锁可用为止， 注意，它不是阻塞的。

**读写锁**(QReadWriteLock)

读写锁(QReadWriteLock): QReadWriterLock 对 “read” , “write”访问进行区别 对待，写线程执行的时候会阻塞所有的读线程，而读线程之间的运行不需要进行 同步 ，使得多线程程序更具有并发性。

QReadWriteLock:: lockForRead() 以读的方式加锁线程

QReadWriteLock:: lockForWrite() 以写的方式加锁线程

QReadLocker 和 QWriteLocker ：是一组方便的类， 可以自动死锁和解锁一个QReadWriteLock类。

**信号量**(QSemaphore)

信号量(Qsemaphore)：也可以被用来使线程的执行顺序化，和QMutex的方法相 似。信号量和互斥量的不同在于，信号量可以在同一时间被多于一个的线程访问。

– QSemaphore::available():

– 返回信号量当前可使用的访问的数量。

– QSemaphore::acquire()

– 试图得到信号量的访问。如果available() == 0，这个调用将会阻塞， 一直到它能够得到访问。

– QSemaphore::release() 释放信号量的访问。这将会唤醒所有等待访问这个信号量的线程。

– QSemaphore::tryAcquire() 尝试申请信号量资源，没有足够资源，该函数会立即返回。(非阻塞)

**条件变量**(QWaitCondition)

条件变量(QWaitCondition):它允许线程在一定条件下唤醒其他线程。

•1 bool wait (QMutex \* mutex,unsigned long time = ULONG\_MAX)

• mutex必须由调用线程进行初锁定 。

• 释放锁定的mutex。

• 在线程对象上等待。

• 等待结束后恢复锁定状态。

•2 void wakeAll() 这将会唤醒所有等待QWaitCondition的线程。这些线程被唤醒的顺序依赖于 操组系统的调度策略，并且不能被控制或预知。

•3 void wakeOne ()

• 这将会唤醒所有等待QWaitCondition的线程中的一个线程。这个被唤醒的线 程依赖于操组系统的调度策略，并且不能被控制或预知。

### 1.2.2 难点和解决办法

一、

问题：修改主窗口图标的时候一直没有成功，操作步骤如下：

1.在中制作一个你想要使用.ico图标文件，然后将它添加到你的工程目录中；

2.在此工程目录（.pro文件所在的目录）中新建一个文本文档，写入IDI\_ICON1 ICON DISCARDABLE "xxx.ico"（xxx代表你的.ico文件名），然后将新建的文本文档保存为后缀为xxx.rc的文件（xxx可随意命名）；

3.在工程目录中的.pro文件中添加代码：RC\_FILE += xxx.rc；

4.重新编译（很重要！）；

原因：之前新建一个工程，都会先运行一次之后再添加图标文件及进行对代码的修改，但是Qt creator中使用MSVC编译器，点击运行好像不会重新编译，导致对图标文件的修改不起作用，所以一直没有修改成功！

解决办法：

1.在新建一个工程的时候，在第一次运行之前就将图标文件添加到工程文件中，并修改好代码，这样就不会有问题；

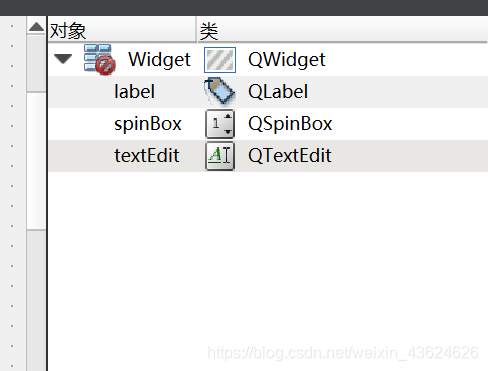
2.在添加好.ico图标文件及修改好代码之后，删掉编译生成的文件即“build-xxx-Desktop\_Qt\_5\_10\_0\_MSVC2015\_64bit-Debug”文件夹，然后再点击运行即可；但是这样做有时候会有问题，当工程中的代码多了以后直接删掉编译生成的文件再次运行时会有一些问题，有隐患，所以这种方法不太建议！

待跟进：在Qt creator 中找到能重新编译的方法，看直接重新编译能否代替删掉编译文件来解决这个问题！

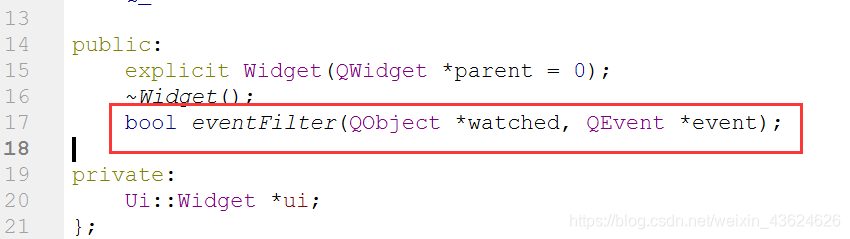
推荐解决办法：每次修改完.pro文件后，先qmake工程，然后再重新构建，最后再运行即可。

### 1.2.3 学习案例

#### 1、新建一个widget，ui界面设计如下

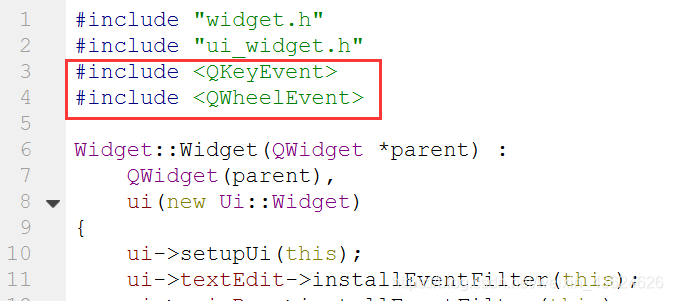


#### 2、widget.h中添加构过滤器造函数

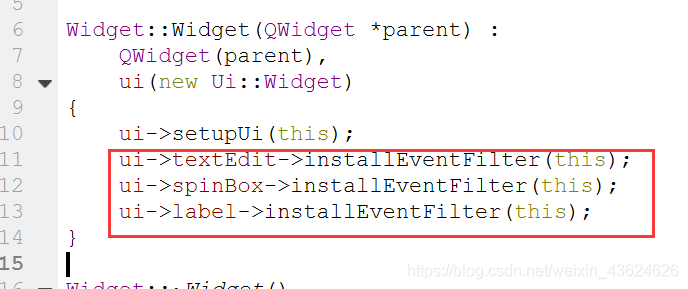


#### 在widget.cpp中添加如下代码

添加头文件

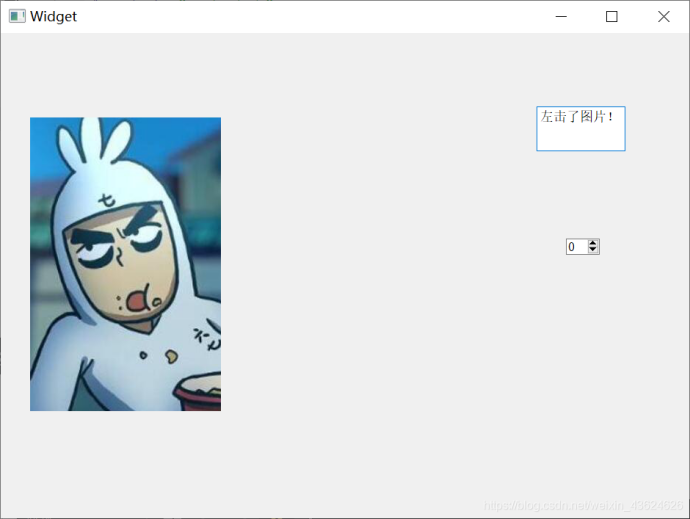


安装过滤器：

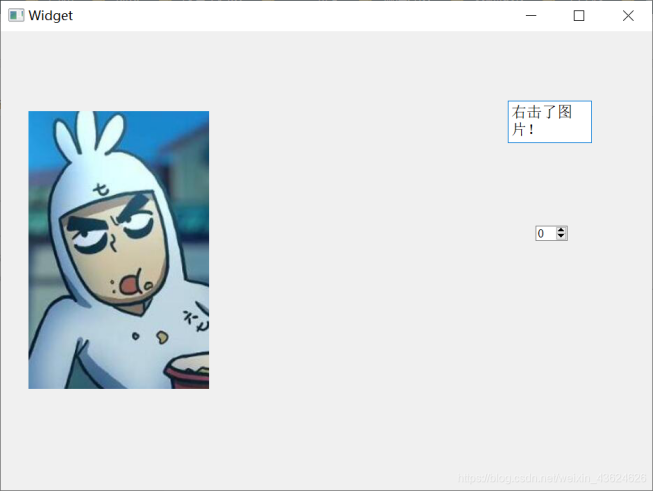


#### 4、运行项目

鼠标左键点击图片



鼠标右键点击图片



鼠标在textEdit上滑动滚轮，改变了字体大小

