基于Qt的桌面日历程序设计及实现

为了介绍具体的设计以及实现过程，首先简要介绍一下功能，接着，将根据功能点，逐项介绍各个功能具体的设计及实现。

功能概要

1. 正确实现按月显示的公历日历。可以参考Qt中`QCalendarWidget`的实现或者直接使用它。
2. 可以编辑日历的某一天：点击某一天可以设置这一天的事件，颜色等属性。
   交互样式可以参考DesktopCal或OneCalendar。
3. 可以添加、编辑、删除某一天的事件序列。例如，某个事件在设置时，可以
   按天、按周、按月或者按其他规律重复发生，要求用户可以直接一次性添加
   一系列事件。要求用户在删除事件时可以选择删除时间序列中的单独一个事
   件或直接删除整个序列的所有事件。
4. 实现本地文件与桌面日历程序的拖拽交互：（1）把本地文件拖入桌面日历程
   序的某一天，在当天的格子内显示文件名称，并且把文件存入桌面日历程序
   的某个文件夹下。（2）可以通过日历程序把存放在某天的文件通过拖拽文件
   名称存放到本地文件夹内进行保存。（3）设置一个按钮用来打开和关闭桌面
   日历程序与本地文件的拖拽交互。
5. 支持使用配置文件（如XML 文件）进行数据同步：支持用户导出所有事件和
   对应颜色数据到配置文件（文本或二进制格式均可）中，并支持程序导入该
   配置文件。
6. 可以对日历进行整体拖拽和固定。参照DesktopCal程序，设置一个按钮可以
   移动和固定整个界面。固定界面之后日历不再响应鼠标事件，并且在透明区
   域让鼠标事件仍然传递到桌面。
7. 国际化：至少支持中文、英文两种模式。
8. 用户数据以及通过拖拽方式保存到日历程序的文件，采用内存数据库（In-Memory
   Database）存储，如采用SQLite 数据库，或其他优秀的内存数据库（如Derby）。
9. 支持对事件和文件名称的全文检索。

显示日历

事实上，样例程序中使用的`QCalendarWidget`完全能够满足这项要求。然而，经过研究，使用QCalendarWidget的话，在处理“文件拖拽”的时候会遇到一定的麻烦。

所以，我从头实现了日历的显示。

首先，我实现了`DateItem`。它继承自`QWidget`，用来存储及显示某一天的日期，显示节日信息、待办事项以及用户存储的文件列表。

我主要重写了`QWidget`的`paintEvent`，在其中使用`QPainter`向屏幕绘制上述内容。其中，待办事项以及用户存储的文件列表是一个HTML字符串，由其他模块根据数据生成，`DateItem`直接将此HTML字符串渲染。日期和节日信息则直接绘制。

接着，一个`QGridLayout`存放了一些按钮用来控制某些功能的行为，以及一些`QLabel`来显示星期几、周号等简单的文本信息，还有6行（一行为一周，一个月最多可能分别占上6周）x 7列（一周七天）共42个`DateItem`真正用于显示和日期相关的信息。

要正确显示日历，就应当知道每一个格子（Grid，也就是`DateItem`）对应的日期。方法十分简单：对于任意给定的月份（包括年和月的信息），首先使用`QDate`内置函数计算出它的第一天是星期几，然后即可计算出当前月份的按月显示的日历的首行首列对应的日期。据此，就可以推算并正确显示出每一个格子的日期了。

编辑某一天，事件序列创建、读取、修改和删除

我将某一天单独的事件以及有规律（每月一次，每周一次等）发生的事件合并，使用统一的结构来存储。每一条事件数据主要分为两个部分，规则以及待办事项。待办事项包括文本和颜色。文本就是一串字符串，保存用户的输入；颜色就是该待办事项文字的背景颜色。规则的颜色指的是规则对应待办事项的颜色。

我这里使用的规则，类似于`crontab`规则的简化版。`crontab`规则在这里就不做说明了，我的简化版包括四个数字和一个字符串，分别是年、月、日、星期几的匹配规则以及排除日期列表。

年、月、日以及星期几的匹配方法是一样的，以年为例，对于某一天，如果这一天的年等于规则的年，或者规则的年等于-1，则认为”年“字段匹配。也就是，-1相当于一种”通配符“。

一个日期与一条规则匹配，当且仅当年、月、日以及星期几四个字段均匹配，并且日期格式化为yyyyMMdd格式的字符串之后不是排除日期列表的子串。

一个日期与一条规则完全匹配，当且仅当日期与规则的年、月以及日三个字段完全相等。

当一个日期能够匹配一条规则，那么这条规则对应的待办事项就属于这一天的待办事项；当一个日期完全（精确地）匹配一条规则，那么这条规则对应的待办事项就属于这一天单独的事件，这条规则的颜色控制着这一天的颜色（当天整个格子的背景颜色）。如果一个日期能够完全匹配多个颜色不同的规则，那么该日期的颜色为数据库中最新插入的一条（能完全匹配该日期的）规则的颜色。

每一条数据的创建、读取、修改和删除操作就对应于SQL的`INSERT`、`SELECT`、`UPDATE`以及`DELETE`语句。具体地，本项目所有SQL语句均在`sqlitestorage.cpp`文件中写明。

在执行SQL语句之前，先用字符串`"QSQLITE"`为参数，调用`QSqlDatabase::addDatabase`创建一个`QSqlDatabase`的实例作为数据库连接。`QSQLITE`表明使用SQLite数据库的驱动程序。然后用数据库路径初始化这个连接。每次执行SQL语句的时候使用`QSqlQuery`类，提供数据库连接、SQL语句即可。必要的时候还需要调用`addBindValue`或`bindValue`来提供SQL语句所需的匿名或具名参数。

我实现了`SQLiteStorage`类，它实现了`TodoStorage`接口，对执行各种SQL语句进行了封装，其他模块只需要调用`add`、`get`、`del`等函数即可在对象层面上存取事件数据。

使用数据库存储各类数据

使用数据库存储待办事项数据在以上功能的说明中已经介绍完毕了。这里主要介绍存储用户文件以及搜索功能的设计和实现方法。

我实现的`SQLiteStorage`类，也实现了`FileStorage`接口。存取文件与存取事件的差别，实际上只体现在数据类型上，文件数据用`BLOB`类型存储，其余操作与事件存取完全一致。

搜索文件名、待办事项的功能，由于时间和知识所限，我没有使用全文索引的办法，而是直接构造一个SQL语句，使用SQL的LIKE操作符来查询数据库。对于用户输入的形如`x y z`的空格分割单词的查询字符串，构造`SELECT * FROM `表名` WHERE (`查询字段` LIKE ?) AND (`查询字段` LIKE ?) AND (`查询字段` LIKE ?)`并将`%x%`、`%y%`和`%z%`作为参数传入，运行该SQL语句即可查到需要的信息了。

拖拽文件处理

拖拽文件实际上是拖拽的文件路径，而不是文件内容。拖拽统一由拖拽目的地程序处理。

拖入和拖出我是分别实现的。

拖入存储

接受鼠标拖入，主要就是设置窗体的`AcceptDrops`为`true`，重写拖动进入事件`dragEnterEvent`、拖动移动事件`dragMoveEvent`以及放置事件`dropEvent`。

这三个事件的参数中都间接携带有`QMimeData`实例，存放着被拖动对象的信息。程序通过判断MIME类型，就可以得知用户拖动的是什么样的数据，从而拒绝或接受用户的拖放请求。拖动文件的MIME类型实际上是`text/uri-list`，URI列表。通过`QMimeData`，程序还可获知具体的URL，从而得知被拖动对象是否是本地路径，是否是文件。如果用户拖动的是URI列表，并且是本地路径，而且不是文件夹，那么程序就接受用户的请求。

在`dropEvent`中接受请求之后，根据用户拖进来的文件路径，将文件通过上述`FileStorage`接口复制到数据库中存储到相应的日期。

为了得知用户拖入的是哪一个日期格子，程序需要使用`dropEvent`中鼠标位置，通过`QWidget`的`childAt`成员函数来获取`DateItem`，从而获取日期。如果用`QCalendarWidget`，就需要自己通过计算（根据`QCalendarWidget`的大小计算）、测量（直接测量每个日期格子的坐标，打表）或偷取（`QCalendarWidget`的`paintCell`中会给出某个`QRect`对应的日期，偷偷存下它）的办法，来获得鼠标位置对应的日期格子了。

拖出读取

拖出相对于拖入要简单一些，通过重写鼠标按下事件`mousePressEvent`来实现。

用户按下鼠标，意味着用户想拖出文件。首先，程序将文件数据从数据库复制到临时目录中相同文件名的文件里。

然后，程序创建一个`QMimeData`实例，设置URI列表为这个文件的绝对路径。

最后，创建一个`QDrag`实例，传入这个`QMimeData`实例，执行`exec`即可向操作系统发出拖动请求。之后，由操作系统来处理文件的复制。

用户数据导出导入

我实现数据库存储用户数据之前，使用的是JSON文件存储用户数据。JSON是一种很好的格式，它轻量级，又同时方便人和机器的解析与生成。所以我决定，导出导入的数据还以JSON的方式存储。

导出，也就是生成JSON，主要使用`QJsonArray`以及`QJsonObject`类。从数据库读出数据，用数据创建出`QJsonObject`或`QJsonArray`对象，之后，将其传给`QJsonDocument`类构造一个JSON文档对象，调用文档对象的`toJson`函数即可获得JSON字符串，写入文件保存即可。

导入的过程和导出相反，从文件读取JSON字符串，创建`QJsonDocument`，获得`QJsonObject`或`QJsonArray`对象，转换成可以存入数据库的格式，写入数据库即可。

拖动窗体

我实现了点击窗体任意非控件的地方都可以拖动窗体的功能。

只需要重写窗体的`mousePressEvent`、`mouseMoveEvent`和`mouseReleaseEvent`事件，并在`mousePressEvent`中记录窗体坐标、鼠标坐标，记录鼠标状态为已经按下；在`mouseMoveEvent`中判断鼠标状态，根据鼠标坐标和`mousePressEvent`记录的两个坐标计算出窗体新坐标并调用`move`来移动窗体；在`mouseReleaseEvent`中记录鼠标状态为已经释放。

对鼠标事件的透明

只需要设置窗体`WindowFlags`加上`Qt::WindowTransparentForInput`即可使得窗体不响应鼠标键盘等输入事件，并传递到"下面"的窗体。

然而，设置了之后，还需要提供一种办法恢复。这里，我使用了一种不太优雅的方法，就是创建一个家长对象为空的与原窗体上切换是否透明的按钮相同大小相同位置的独立的`QPushButton`盖在上面，用来控制窗体恢复。

另外，经过测试发现，`WindowTransparentForInput`在Windows（Windows 10 x64）和Linux（Ubuntu，Unity）系统上表现十分良好，可以动态切换，而在OS X下似乎只能在窗口初始化前设置好才有效。

国际化

国际化要在项目一开始就注意，否则之后找出所有字符串并修改会比较困难。国际化的实现，就是使用`tr()`包围要显示出来的字符串。每次添加或修改`tr`包裹的字符串之后，运行`lupdate`来生成新的`.ts`文件。然后，使用任意一种文本编辑器或者使用Qt语言家来进行手动翻译。最后，运行`lrelease`来生成经过优化、压缩的二进制语言包`.qm`文件。

将语言包加入资源后，在主程序中使用`QTranslator`加载语言包并安装`QTranslator`实例到`QApplication`就可以了。

我使用`QLocale`来检测操作系统的语言，以此来决定界面显示中文的还是英文的。

在Mac OS X或Linux下，可以在启动程序之前临时修改环境变量`LANG=en_US.UTF8`或`LANGUAGE=en_US`来启动英文界面的日历。