QCalendar 设计文档

THU CST55 王逸松

1 Overview

QCalendar 是一个基于 Qt 的桌面日历程序。

该程序可以对日历的每一天，设置颜色和事件，所有事件也将会以倒计时的形式显示。

事件可以是发生一次的，或者是按一定规律（如，每 3 天，或者每 1 周）重复的。

该程序的窗口可以整体移动，或者固定在桌面上，并设置鼠标点击穿透。

配置可以通过文件进行导入导出，从而实现数据同步。

由于使用 Qt 进行开发，该程序可以运行在各种桌面系统上（如 Windows, Linux, OS X）。

2 System Architecture

该程序主要分为五个部分，**主窗体**、**日历模块**、**事件模块**、**事件对话框**、**设置对话框**。

2.1 主窗体

主窗体是一个 QWidget，内部放置了一个 QGridLayout 用来显示日历模块和事件模块。

主窗体可以控制日历模块和事件模块，具体来说，可以控制他们显示的内容，以及处理他们的操作。

当主窗体的长大于宽的时候，两个模块会被横屏显示，否则会竖屏显示。

2.2 日历模块

日历模块是一个 QWidget，内部放置了一个 QGridLayout，里面有几十个 QLabel，用来显示每一天。

这里没有使用到 Qt 自带的 QCalendarWidget，而是直接用 QLabel 来实现的。

当某个 QLabel 被点击的时候，这个 label 对应的那一天会被设置为被选中，如果原来就被选中，那么会被取消选择。在这之后，主窗体会被通知（通过 Qt 的 signal-slot 机制），接下来事件模块会显示有关这一天的具体信息。

每一天的颜色可以被设置，这是在事件模块中实现的。设置好之后相应的颜色会直接显示在日历上这一天的格子里（这是通过 QLabel 的 FrameStyle 来实现的）。

2.3 事件模块

事件模块分为两部分，一部分是 UI，一部分是**事件管理器**。

2.3.1 事件模块 UI

事件模块 UI 是一个 QWidget，内部用一个 QGridLayout 来管理布局。

当日历模块中没有选择任何一天的时候，该 UI 会显示当前时间，以及即将到来的几个事件。如果没有足够的事件，会再显示已经过去的事件，按照时间倒序。

当日历模块中已经选择了某一天的时候，该 UI 会显示这一天的事件，以及这一天的颜色选项（可以通过这个选项来设置日历中这一天的颜色）。

点击某个事件，或者点击“添加事件”按钮，会弹出一个事件对话框，可以添加/修改事件。

另外该 UI 上还有一个设置按钮，可以打开设置对话框。

2.3.2 事件管理器

事件管理器是一个 Class，用来管理所有事件。

事件管理器支持的操作有添加事件、删除事件、修改事件、查询一个时间区间内的所有事件、查询一个时间点之前/之后的前 x 个事件等。

事件模块 UI 需要跟事件管理器交互，来得到每天的具体事件信息。

事件对话框在添加/修改/删除事件的时候，也需要跟事件管理器交互。

2.4 事件对话框

一共有两个事件对话框，都是用 QDialog 实现，不妨记为事件对话框 A 和 B。

2.4.1 事件对话框 A

事件对话框 A 是用来显示事件的具体信息的，当在事件模块 UI 中点击了某个事件的时候会被弹出。

对话框中有“编辑”和“删除”两个按钮，可以对其显示的事件进行相应的操作。

2.4.2 事件对话框 B

事件对话框 B 是用来编辑和添加事件的，当在事件对话框 A 中点击了“编辑”，或在事件模块 UI 中点击了“添加事件”的时候会被弹出。

每个事件有事件名称、时间、地点、重复周期这些选项，其中地点非必填。

2.5 设置对话框

设置对话框也是一个 QDialog，用来管理整个应用程序的设置。

设置有透明度、语言。通过一个 QSilder 可以调节主窗体的透明度，通过两个 QRadioBox 可以调节整个应用程序的语言（英文/中文）。

另外还可以将所有设置、颜色选项、事件保存到一个文件，或者从文件读取这些内容。

3 Data Design

这里主要讲颜色和事件，以及保存到文件的方式。

3.1 颜色

颜色在程序中使用 QColor 来保存。

在需要显示的时候，QColor 会被转成形如“#ffffff”的字符串，然后设置到 label 的 FrameStyle 上。

在主窗体和日历模块中，每一天的颜色被存入一个 QMap<QPair<int, QColor> >中，其中 int 为将那一天的日期转成 int（具体来说，Y * 32 * 13 + M * 32 + D）之后的值。

3.2 事件

事件在程序中使用一个自定义的结构体 MyEvent 来保存。

其中 MyEvent 的定义为：

```
struct MyEvent {
  long long ID;
  long long datetime;
  QString name;
  QString location;
  int repeat_type; // 0 -- no, 1 -- day, 2 -- week, 3 -- month, 4 -- year
  int repeat_interval;
};
```

如上，ID 为一个随机的整数，保证每个事件的 ID 都不相同。

datetime 为事件发生的时间转成 long long 之后的值，具体来说，为 (Y * 32 * 13 + M * 32 + D) * 86400 + 时间发生的时刻在当天的秒数。

name 和 location 分别为事件的名称和地点。

repeat_type 和 repeat_interval 为重复的方式和重复间隔。

在事件管理器中，所有事件被放入了一个 QMap 中，下标按照事件的 ID 排序，于是能快速找到某个 ID 对应的时间。

另外有一个 QMap，按时间顺序储存了所有事件，从而查询某时间点前后的事件可以实现。

还有一个 QMap，按时间顺序储存了所有**会重复发生的**事件。每一秒钟，事件管理器会利用这个 QMap 检查哪些事件刚刚发生，然后给他们设置下一次发生的时间。

当需要添加/修改/删除事件的时候，事件管理器可能正在进行其他操作，这容易导致程序运行不正确，所以整个事件管理器的加了锁，任何操作前都需要先检查锁的状态（这里并没有用任何库函数实现，但是出错率已经能降低到 10^{-9} 的数量级）。

3.3 保存到文件

设置、颜色、事件都会被保存到文件。

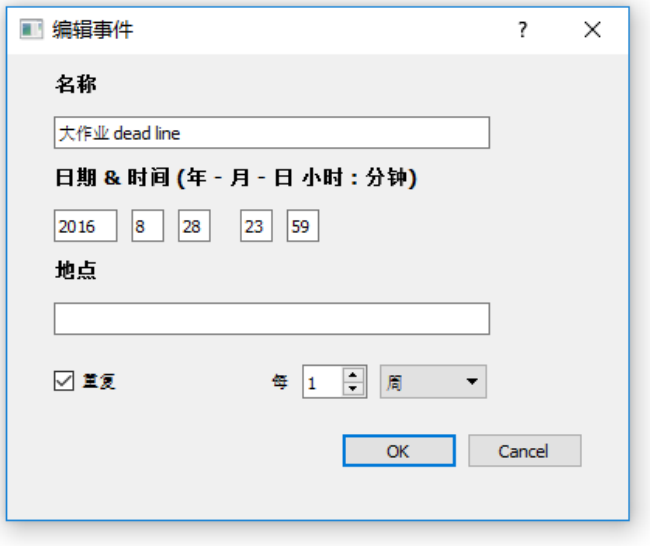
这个文件是一个二进制文件，即使用 QDataStream 来实现文件 IO。

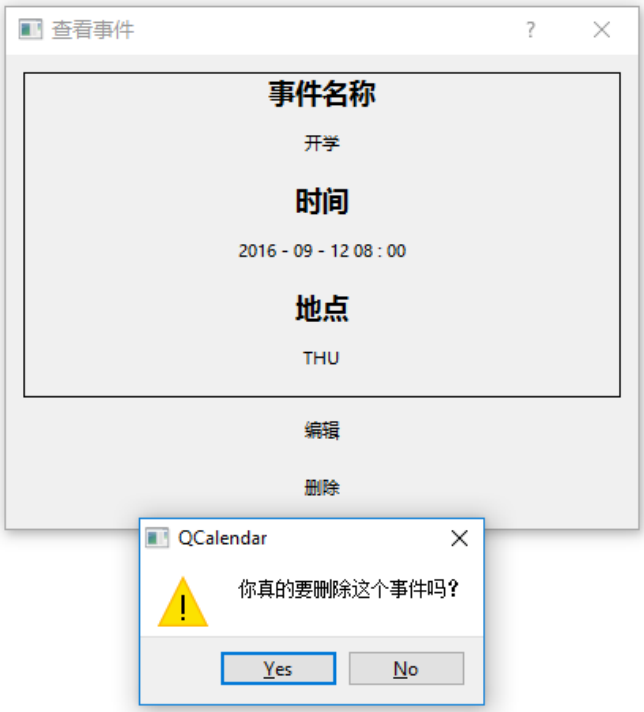
保存设置的方法是，将语言和透明度作为 int 直接输出。

保存颜色的方法是，先输出不是白色的天数（因为默认颜色是白色），然后输出每一天的日期（int 格式）和颜色（字符串格式）。

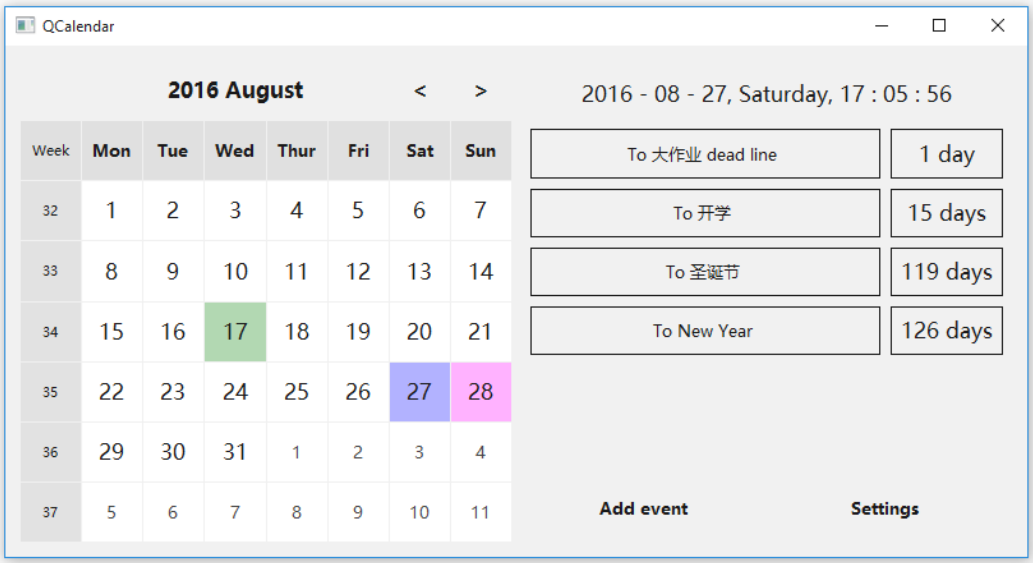
保存事件的方法是，先输出事件的总个数，然后输出每个事件的所有信息。

4 Screenshots


编辑事件
名称
大作业 dead line
日期 & 时间 (年 - 月 - 日 小时：分钟)
2016
8
28
23
59
地点
重复
每
1
周
OK
Cancel


查看事件
事件名称
开学
时间
2016 - 09 - 12 08 : 00
地点
THU
编辑
删除
QCalendar
你真的要删除这个事件吗?
Yes
No


设置
透明度
Language/语言
English/英文
Chinese/中文
保存配置到文件
从文件载入配置


QCalendar
2016 August
<
>
2016 - 08 - 27, Saturday, 17 : 05 : 56
Week
Mon
Tue
Wed
Thur
Fri
Sat
Sun
32 1 2 3 4 5 6 7
33 8 9 10 11 12 13 14
34 15 16 17 18 19 20 21
35 22 23 24 25 26 27 28
36 29 30 31 1 2 3 4
37 5 6 7 8 9 10 11
To 大作业 dead line
1 day
To 开学
15 days
To 圣诞节
119 days
To New Year
126 days
Add event
Settings