# QSettings 读写注册表、配置文件

# 简述

一般情况下，我们在开发软件过程中，都会缓存一些信息到本地，可以使用轻量级数据库sqlite，也可以操作注册表、读写配置文件。

关于QSettings的使用前面已经介绍过了，比较详细，见**“更多参考”**，下面介绍下QSettings常用功能-读写注册表、配置文件。

# 优点

**无需指定注册表路径**

一般情况下，我们需要定义一个宏，或者常量字符串来指定保存的注册表位置。

* #define HKEY\_CURRENT\_USER\_QT "HKEY\_CURRENT\_USER\\SoftWare\\Digia\\Qt"
* const QString HKEY\_CURRENT\_USER\_QT = "HKEY\_CURRENT\_USER\\SoftWare\\Digia\\Qt";

**无需指定配置文件路径**

一般情况下，我们需要定义一个宏，或者常量字符串来指定保存的配置文件位置及名称。

* #define INI\_QT "C:\Users\WangLiang\AppData\Roaming\Digia"
* const QString INI\_QT = "C:\Users\WangLiang\AppData\Roaming\Digia";

采用下面方式，我们不需要做太多工作，Qt已经很好的替你实现了！

# 读写注册表

## 一般存储

下面我们以Qt为例，众所周知现在Qt已经属于Digia，也就是说：组织名为Digia，产品名为Qt。

在main()函数中，首先设置组织名、产品名。

QCoreApplication::setOrganizationName(QString("Digia"));

QCoreApplication::setApplicationName(QString("Qt"));

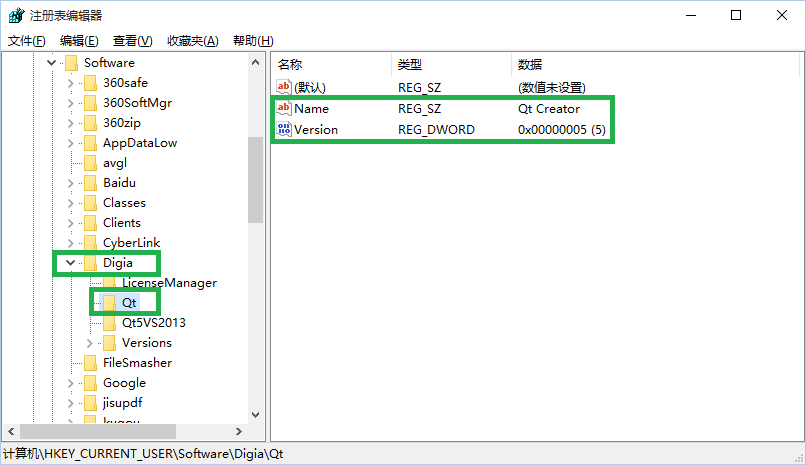
然后使用QSettings对注册表进行操作：

QSettings settings(QSettings::NativeFormat, QSettings::UserScope, QCoreApplication::organizationName(), QCoreApplication::applicationName());

settings.setValue("Name", "Qt Creator");

settings.setValue("Version", 5);

这时，我们打开注册表regedit，数据就生成了。



## 一般读取

存储完数据之后，默认的程序启动时需要加载对应的数据。

QString strName = settings.value("Name").toString();

int nVersion = settings.value("Version").toInt();

*//Name:Qt Creator Version:5*

这时，我们可以通过查看应用程序输出窗口得到输出结果。

## 分目录存储

如果我们需要在同一路径下建立多个子目录该怎么办，下面介绍两种方式。

### 替换applicationName

如上，我们可以看出，organizationName对应的注册表路径为HKEY\_CURRENT\_USER\\SoftWare\\Digia，applicationName对应的为其下一级的目录，那么分目录就需要更改其对应的applicationName。

QSettings settings(QSettings::NativeFormat, QSettings::UserScope, QString("%1\\%2").arg(QCoreApplication::organizationName()).arg(QCoreApplication::applicationName()), "Qt5.5")*;*

settings.setValue("Name", "Qt Creator")*;*

settings.setValue("Version", "5.5")*;*

QSettings settings2(QString("%1\\%2").arg(QCoreApplication::organizationName()).arg(QCoreApplication::applicationName()), "Qt5.6")*;*

settings2.setValue("Name", "Qt Creator")*;*

settings2.setValue("Version", "5.6")*;*

### 分组

替换applicationName的方式看起来有些繁琐，相比之下，使用group分组则会更简单！

QSettings settings*;*

settings.beginGroup("Qt5.5")*;*

settings.setValue("Name", "Qt Creator")*;*

settings.setValue("Version", "5.5")*;*

settings.endGroup()*;*

settings.beginGroup("Qt5.6")*;*

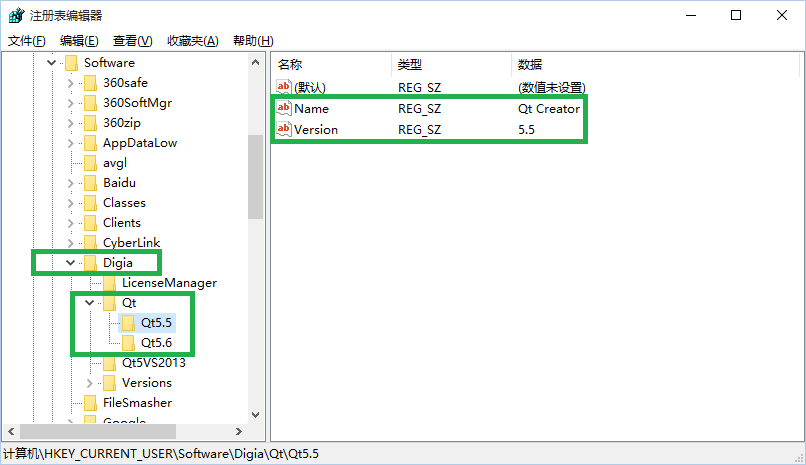
settings.setValue("Name", "Qt Creator")*;*

settings.setValue("Version", "5.6")*;*

settings.endGroup()*;*

这时，我们再次查看注册表数据。

**注：**   
新建目录，则需要重新打开注册表，如果新添加设置，则不需要重新打开注册表，只需要来回切换对应的选项即可。



# 读写配置文件

## 一般存储

如上，我们只需要将格式从NativeFormat变为IniFormat即可：

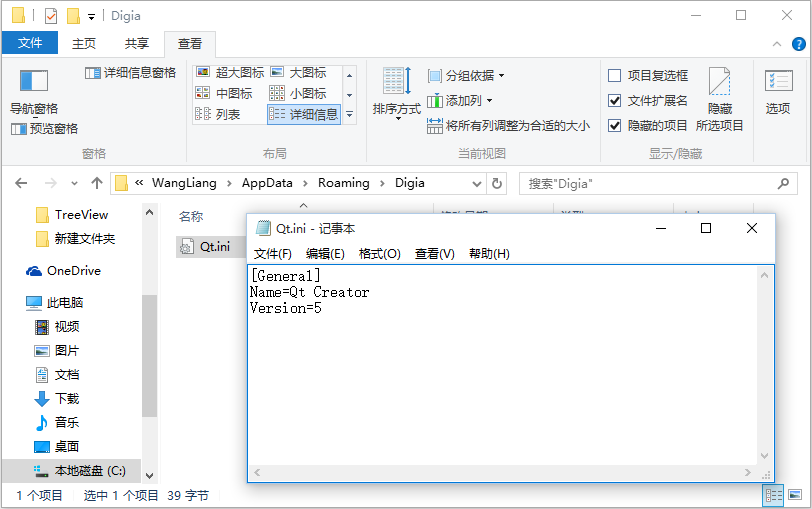
QSettings settings(QSettings::IniFormat, QSettings::UserScope, QCoreApplication::organizationName(), QCoreApplication::applicationName());

settings.setValue("Name", "Qt Creator");

settings.setValue("Version", 5);

这时，我们打开对应的存储目录，数据就生成了。

我们可以进入文件夹：C:\Users\WangLiang\AppData\Roaming（AppData默认为隐藏文件，需要设置显示才可查看），可以看到生成了文件夹”Digia”以及配置文件”Qt.ini”。



## 一般读取

存储完数据之后，默认的程序启动时需要加载对应的数据。

QString strName = settings.value("Name").toString();

int nVersion = settings.value("Version").toInt();

*//Name:Qt Creator Version:5*

这时，我们可以通过查看应用程序输出窗口得到输出结果。

## 分组存储

我们可以看到配置文件中包含默认的分组为：General。通常情况下，我们需要对配置进行归类，例如：用户名、密码等信息属于用户组，产品名称、版本号属于设置组。

QSettings settings(QSettings::IniFormat, QSettings::UserScope, QCoreApplication::organizationName(), QCoreApplication::applicationName())*;*

settings.beginGroup("Setting")*;*

settings.setValue("Name", "Qt Creator")*;*

settings.setValue("Version", 5)*;*

settings.endGroup()*;*

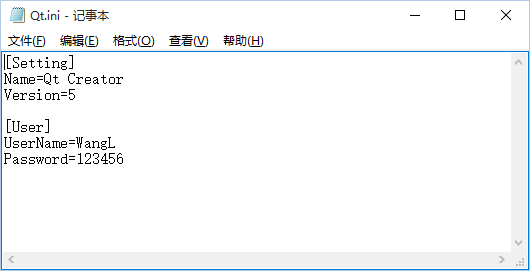
settings.beginGroup("User")*;*

settings.setValue("UserName", "WangL")*;*

settings.setValue("Password", "123456")*;*

settings.endGroup()*;*

这时我们再次查看配置文件，里面已经生成了另外两个分组。



## 分组读取

settings.beginGroup("Setting")*;*

QString strName = settings.value("Name").toString()*;*

int nVersion = settings.value("Version").toInt()*;*

settings.endGroup()*;*

//Name:Qt Creator Version:5

settings.beginGroup("User")*;*

QString strUserName = settings.value("UserName").toString()*;*

QString strPassword = settings.value("Password").toString()*;*

settings.endGroup()*;*

//UserName:WangL Password:123456

## 分目录存储

什么时候需要分目录存储呢？QQ大家都用过吧，是不是每一个用户都有一个对应QQ号的目录呢，里面保存各个用户对应的信息。

如上，我们可以看出，organizationName对应的本地路径为C:\Users\WangLiang\AppData\Roaming\Digia，applicationName对应的为其下的配置文件，那么如果我们需要在同一路径下建立多个目录就需要更改对应的organizationName，配置文件名称则需要更改其对应的applicationName了。

QSettings settings(QSettings::IniFormat, QSettings::UserScope, QString("%1\\%2\\%3").arg(QCoreApplication::organizationName()).arg(QCoreApplication::applicationName()).arg("Qt5.5"), "User")*;*

settings.setValue("Name", "Qt Creator")*;*

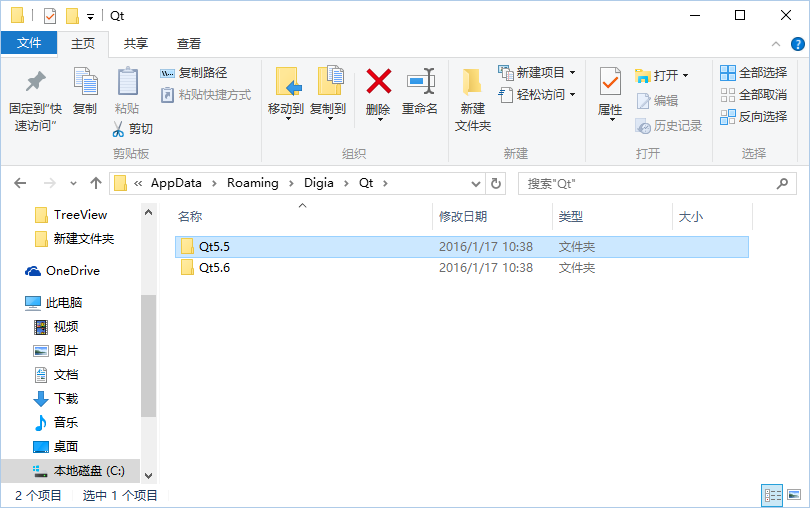
settings.setValue("Version", "5.5")*;*

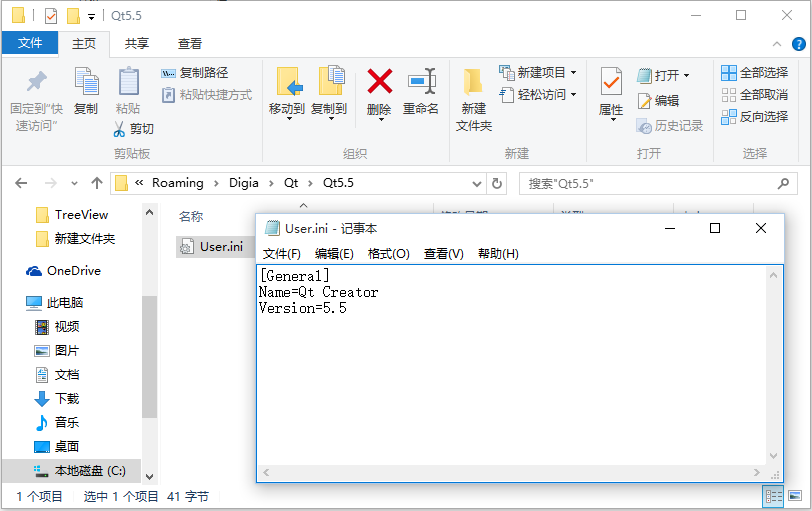
QSettings settings2(QSettings::IniFormat, QSettings::UserScope, QString("%1\\%2\\%3").arg(QCoreApplication::organizationName()).arg(QCoreApplication::applicationName()).arg("Qt5.5"), "User")*;*

settings2.setValue("Name", "Qt Creator")*;*

settings2.setValue("Version", "5.6")*;*

这时，我们再次查看本地文件，则会发现C:\Users\WangLiang\AppData\Roaming\Digia\Qt所在目录下会生成两个文件夹”Qt5.5”和”Qt5.6”，并且每一个目录底下会生成对应的配置文件User.ini。





# 删除内容

删除一个指定的键

QSettings settings*;*

settings.setValue("Name", "Qt Creator")*;*

settings.setValue("Version", 5)*;*

settings.remove("Name")*;*

QStringList keys = settings.allKeys()*;*

// keys: ["Version"]

清空所有键

settings.clear();

QStringList keys = settings.allKeys();

// keys: []

删除设置键以及子设置键

QSettings settings*;*

settings.setValue("Qt5.6", "5.6")*;*

settings.beginGroup("Qt5.5")*;*

settings.setValue("Name", "Qt Creator")*;*

settings.setValue("Version", "5.5")*;*

settings.endGroup()*;*

settings.beginGroup("Qt5.6")*;*

settings.setValue("Name", "Qt Creator")*;*

settings.setValue("Version", "5.6")*;*

settings.endGroup()*;*

settings.remove("Qt5.6")*;*

QStringList strList = settings.allKeys()*;*

// keys: ["Qt5.5/Name", "Qt5.5/Version"]

如果key为空字符串，在当前group()的所有键将被删除。

QSettings settings*;*

settings.setValue("Qt5.6", "5.6")*;*

settings.beginGroup("Qt5.5")*;*

settings.setValue("Name", "Qt Creator")*;*

settings.setValue("Version", "5.5")*;*

settings.endGroup()*;*

settings.beginGroup("Qt5.6")*;*

settings.setValue("Name", "Qt Creator")*;*

settings.setValue("Version", "5.6")*;*

settings.endGroup()*;*

settings.beginGroup("Qt5.6")*;*

settings.remove("")*;*

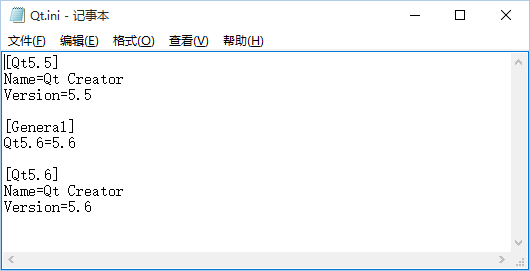
settings.endGroup()*;*

QStringList keys = settings.allKeys()*;*

// keys: ["Qt5.5/Name", "Qt5.5/Version"]

# 疑问解释

如上文代码，我们可以知道未删除之前keys: ["Qt5.6", "Qt5.5/Name", "Qt5.5/Version", "Qt5.6/Name", "Qt5.6/Version"]，其中Qt5.6所在分组为默认的General。



那么调用下面代码：

settings.beginGroup("Qt5.6")*;*

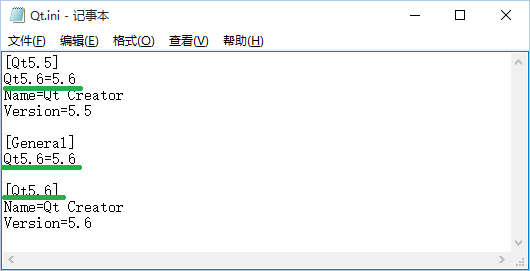
settings.remove("")*;*

settings.endGroup()*;*

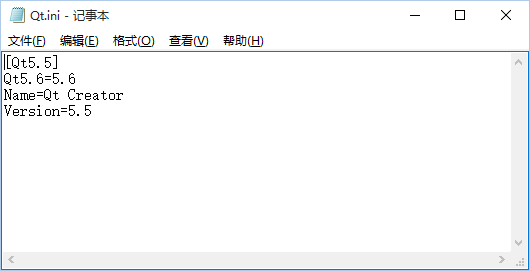
应该只删除对应组中的所有键才对，也就是说剩余的keys应该为 ["Qt5.6", "Qt5.5/Name", "Qt5.5/Version"]，General下的键为什么会被删除呢？好，这里先跳过，继续。

General下的键既然可以被删除，那么在Qt5.5分组下建立对应的Qt5.6键值，应该也会被删除。

删除前：



删除后：



为什么Qt5.5分组下的Qt5.6对应的键还在呢？

我们继续分析：

删除前：keys: ["Qt5.6", "Qt5.5/Qt5.6", "Qt5.5/Name", "Qt5.5/Version", "Qt5.6/Name", "Qt5.6/Version"]。

助手中关于remove()的说明为：Removes the setting key and any sub-settings of key.

也就是说：Qt5.5/Qt5.6键中即使存在Qt5.6，但是所属的setting key为Qt5.5而非Qt5.6，所以不会被删掉。

既然这样，那么我们的疑问也就不复存在了。