第3章　Qt简介

这一章将会对用于本书其余部分的一些风格指南和命名约定进行介绍。Qt核心模块会用一些例子和使用了Qt流及数据类的练习中进行介绍。

Qt是一个类和工具的模块化系统，它可以使用户更容易地编写出自己的小代码模块。Qt提供了一个近乎完美的STL类/类型替代品，可在比使用C++0x编写的代码更多的编译器上进行构建/运行，并支持一些无须现代编译器的同样特性。这一章将介绍如何开始复用Qt。

在Qt的源文档内，有一个示例和范例(Examples and Demos)集，有时会需要引用它，它们位于带有“examples/”或者“demos/”路径前缀的目录中。如果使用的是Qt SDK或者是用Linux安装的Qt，或许需要运行安装包管理器来安装它们。图3.1给出了来自Qt Creator欢迎画面的Getting Started界面。

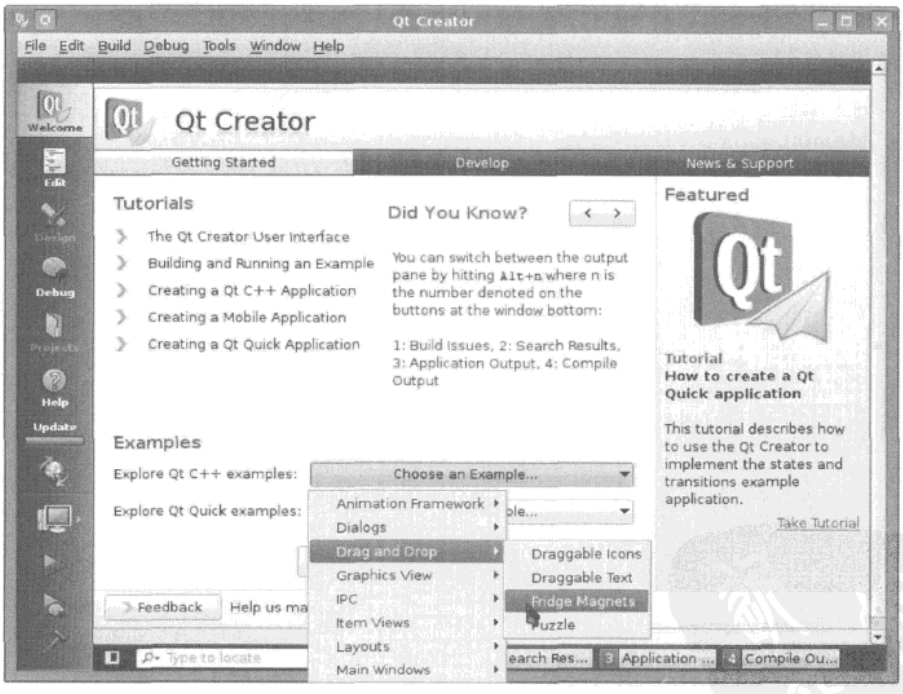

图3.1　Qt Creator的欢迎界面

注意

附录E中给出了一些在不同平台上进行快速设置的提示。

3.1　风格指南与命名约定

C++是一种支持多种不同编程风格的功能强大的编程语言。在大多数 Qt 程序中使用的编码风格并非“纯”C++，取而代之的是一种宏与预处理技巧相结合的风格，以此来获得一种相对于 C++来说更加类似于 Java 和 Python 的高级动态语言。事实上，为了更充分地利用 Qt 的强大功能和使用简单的特性，我们倾向于完全放弃标准库(Standard Library)。

在任意协作严密的编程项目中，可能都有风格指南来提高编写的代码的可读性、可复用性和可靠性。在[qtapistyle]和[kdestyle]中给出了半官方的 Qt 编程风格指南。这里对采用的一些风格总结如下。

- 名称是一些字母和数字构成的序列，第一位不能为数字。
- 名称的第一位也可以使用下画线字符(_)，但是不鼓励使用它，除非是用于类的数据成员。
- 类名称以大写字母开头，例如：`class Customer`。
- 函数名称以小写字母开头。
- 通过合并多个单词并且让每个单词首字母大写，即用“驼峰规则”(CamelCase)的方式构造多单词的名称，例如，`class MetaDataLoader`,`void getStudentInfo()`。
- 常量应当大写并且尽可能在类的作用域内创建成枚举值，全局常量和宏通常应当都是全部大写。
- 每一个类名称都应当是一个名词或者名词短语，例如，`class LargeFurryMammal`。
- 每一个函数名称都应当是一个动词或者动词短语，例如，`processBookOrder()`。
- 用于 `if()`语句时，每一个布尔变量都应当近似于一个句子，例如，`bool isQualified`。

对于数据成员，本书采用改进的匈牙利标记法，其中会使用共同的前缀，以便在代码中让数据成员总是清晰可见。

- 数据成员：`m_Color`，`m_Width`——以小写字母 m 开头。
- 静态数据成员：`s_Singleton`，`s_ObjCount`——以小写字母 s 开头。

对于每一个属性，其相应的获取器/设置器(getter/setter)都有约定俗成的命名规则。

- 非布尔型获取器：`color()`或者 `getColor()`[①]。
- 布尔型获取器：`isChecked()`或者 `isValid()`。
- 设置器：`setColor(const Color& newColor)`。

一致的命名约定往往能够极大地提高程序的可读性和可维护性。

3.1.1　其他编码标准

下面是一些广泛应用的其他编码标准。需要记住的是，与 Qt 编程最为相关的风格依旧是[qtapistyle]。

① 后者属于 Java 风格，而前者属于 Qt 风格。两种约定都有广泛的使用，只要在代码中保持一致即可(本书中并没有保持一致，是因为希望能够展示一些不同的约定)。

- C 和 C++风格文档[①]。
- 编码标准生成器[②]。

3.2 Qt 核心模块

Qt 是一个大库，由数个较小的库或者模块组成，其中最为常见的有如下这些。

- core——包括 QObject, QThread, QFile, Qvariant, 等等。
- gui——所有从 QWidget 派生的类外加一些相关的类。
- xml——用于解析和序列化 XML。
- sql——用于与 SQL 数据库通信。
- phonon——用于播放多媒体文件。
- webkit——用于使用一种嵌入式 Web 浏览器，QtWebkit。

除了 core 和 gui，这些模块都需要在 qmake 的工程文件中启用。例如

```
QT += xml  # to use the xml module
QT -= gui  # to not use QWidgets
QT += sql  # to use SQL module
```

3.2.1 节将会介绍一些核心库中的类。

3.2.1 流和日期

在之前的数个例子中，已经看到 QTextStream 的几个实例，该类在行为上类似于 C++标准库中的全局 iostream 对象。当使用它们来与标准输入(键盘)和标准输出(屏幕)交互时，希望将它们命名成类似的名称，cin, cout 和 cerr。为了使用方便，将这些定义和另外一些有用的函数一起放置到了一个命名空间内，这样就可以轻松地用#include 将这些定义包含到任何程序中了。

示例 3.1 src/qstd/qstd.h

```
[ . . . . ]
namespace qstd {

    // declared but not defined:
    extern QTextStream cout;
    extern QTextStream cin;
    extern QTextStream cerr;

    // function declarations:
    bool yes(QString yesNoQuestion);
    bool more(QString prompt);
    int promptInt(int base = 10);
    double promptDouble();
    void promptOutputFile(QFile& outfile);
```

① 参见http://www.chris-lott.org/resources/cstyle/。

② 参见http://www.rosvall.ie/CSG/。

```
    void promptInputFile(QFile& infile);
};
[ . . . . ]
```

示例 3.1 声明了类似于 iostream 的 QTextStream 对象，示例 3.2 包含了这些静态对象所需的定义。

示例 3.2　src/qstd/qstd.cpp

```
[ . . . . ]

QTextStream qstd::cout(stdout, QIODevice::WriteOnly);
QTextStream qstd::cin(stdin, QIODevice::ReadOnly);
QTextStream qstd::cerr(stderr, QIODevice::WriteOnly);

/* Namespace members are like static class members */
bool qstd::yes(QString question) {
    QString ans;
    cout << QString(" %1 [y/n]? ").arg(question);
    cout.flush();
    ans = cin.readLine();
    return (ans.startsWith("Y", Qt::CaseInsensitive));
}
```

QTextStream 能够针对 Unicode 的 QString 和其他 Qt 类型起作用，因此在后面的示例中，将使用 QTextStream 而不再使用 iostream。示例 3.3 使用 QTextStream 对象和刚刚介绍过的 qstd 命名空间中的函数。该例还使用 QDate 类的一些函数成员以几种不同的格式输出日期。

示例 3.3　src/qtio/qtio-demo.cpp

```
[ . . . . ]
#include <qstd.h>

int main() {
    using namespace qstd;
    QDate d1(2002, 4,1), d2(QDate::currentDate());
    int days;
    cout << "The first date is: " << d1.toString()
            << "\nToday's date is: "
            << d2.toString("ddd MMMM d, yyyy")<< endl;

    if (d1 <  d2)
        cout << d1.toString("MM/dd/yy") << " is earlier than "
                << d2.toString("yyyyMMdd") << endl;

    cout << "There are " << d1.daysTo(d2)
            << " days between "
            << d1.toString("MMM dd, yyyy") << " and "
            << d2.toString(Qt::ISODate)  << endl;

    cout << "Enter number of days to add to the first date: "
            <<  flush;
```

```
    days = promptInt();
    cout << "The first date was " << d1.toString()
            << "\nThe computed date is "
            <<  d1.addDays(days).toString() << endl;
    cout << "First date displayed in longer format: "
            << d1.toString("dddd, MMMM dd, yyyy")  << endl;
[ . . . . ]
```

从这个例子所在的 src 目录，可以构建并运行这个程序。示例 3.4 中给出的工程文件会使用相对路径找到 qstd 的头文件和实现文件。

示例 3.4 src/qtio/qtio.pro

```
CONFIG += debug console
DEFINES += QT_NOTHREAD_DEBUG

CONFIG -= moc
INCLUDEPATH += . ../qstd
DEPENDPATH += ../qstd

# Input
SOURCES += qtio-demo.cpp qstd.cpp
HEADERS += qstd.h
```

以下是程序的输出结果。

```
The first date is: Mon Apr 1 2002
Today's date is: Wed January 4, 2006
04/01/02 is earlier than 20060104
There are 1374 days between Apr 01, 2002 and 2006-01-04
Enter number of days to add to the first date: : 1234
The first date was Mon Apr 1 2002
The computed date is Wed Aug 17 2005
First date displayed in longer format: Monday, April 01, 2002
```

3.3 Qt Creator，用于 Qt 编程的集成开发环境

Qt Creator 是一个跨平台的集成开发环境(Integrated Development Environment，IDE)，它用于简化基于 Qt 的 C++应用程序开发工作。正如所期望的，Qt Creator 含有一个优良的 C++代码编辑器，可以提供智能代码补全、基于 Qt 助手(Qt Assistant)的上下文关联帮助、错误/警告信息键入以及快速代码工具导航等功能。此外，可使用拖动/放下窗体布局来完成设计的 Qt 设计师也完全集成到了 Qt Creator 中。Qt Creator 拥有一个可视化的调试工具和工程构建/管理工具，还有一些非常容易使用的代码生成和导航特性。在 YouTube 上可以获得一些 Qt Creator 的视频教程[①]。

如果仍希望使用自己喜欢的 IDE，并且碰巧是 xcode, MS dev studio 或者 Eclipse，那么你很幸运——可以下载 Qt 集成开发包，它会让你所喜好的 IDE 提供一个与 Qt Creator 相似的特性。从 qt.nokia.com 通过下载二进制安装包或者源代码包，可以把 Qt Creator 安装到 Windows, Mac 和 Linux 平台上[②]。在 Qt 软件开发包(Qt Software Development Kit，SDK)中包含有 Qt Creator，其中包含了 Qt 和快速开始用 Qt 进行开发工作的所有东西。

① 参见http://www.youtube.com/view_play_list?p=22E601663DAF3A14。

② 参见http://qt.nokia.com/downloads。

Qt Creator 用到的项目文件是 qmake 的.pro 文件。通过创建或编辑 Qt Creator 中已有的项目文件，可以完全避免命令行工具的使用。

默认情况下，Qt Creator 会导入工程并会在一个“影子”(shadow)构建目录下建立工程。这就意味着，中间文件和可执行文件并没有和源文件放在一起，而是在其他的地方。而且，程序默认情况下会从那个目录中运行。

要查看/修改工程的构建目录或者运行目录，可以选择 Projects 模式(见图 3.2)并单击 Build 标签修改 Qt 的版本、构建目录或者其他设置。单击 Run 标签，可修改 Run 目录或者其他与运行相关的设置项。

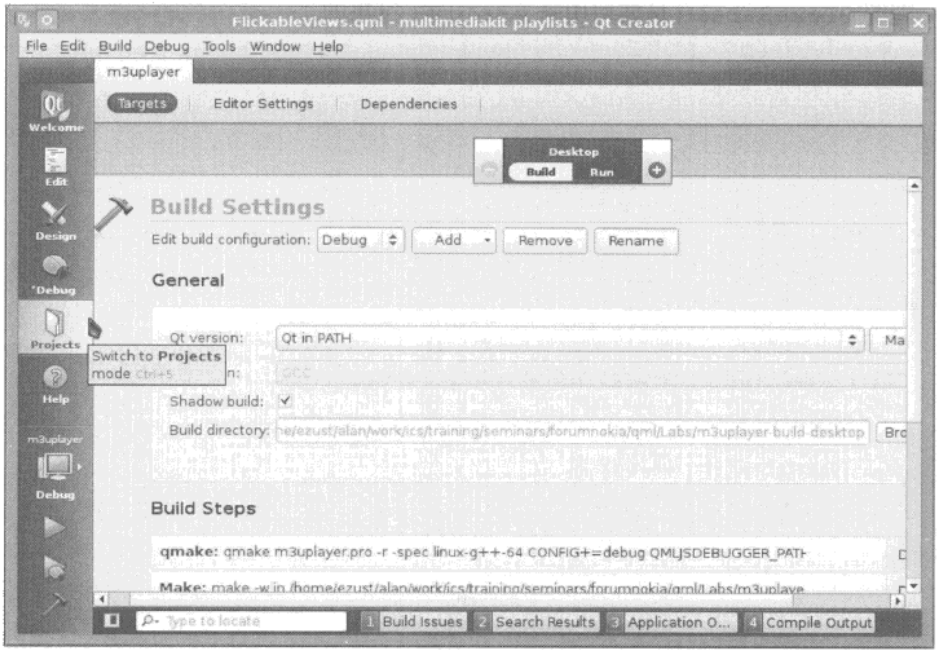

图 3.2　Qt Creator 的 Projects 模式

3.4　练习：Qt 简介

1. 编写一个燃料耗用计算器，可用方向键在英里/加仑和升/100 km 之间转换。
   例如，汽车可以用 1 加仑(美国单位制)燃料行驶 34 英里，那么行驶 100 km 将会耗用 6.92 升的燃料。
   可以使用 1.11 节中演示过的 QInputDialog 和 QMessageBox。
2. 询问用户的生日并计算他的当前年龄。提示：可使用 QInputDialog::getText()把结果从 QString 转换成 QDate，并将其与 QDate::currentDate()比较。或者，可以创建一个对话框，显示一个 QDateEdit，从用户那里得到合适的日期值。

3.5 复习题

1. 什么是 QTextStream? 如何复用它?
2. qmake 工程文件中 Qt 变量的作用是什么? 它可能的取值是什么?
3. 为什么程序中应当使用 QTextStream 而不是使用 iostream?

3.5.1 更多探讨

1. 访问 Unicode 编码的 Web 站点，并解释什么是 Unicode 以及为什么说 QString 支持 Unicode 标准非常重要。
2. 浏览 QTextStream 文档，并编写几个测试从文本文件读取数据方式的小程序。
3. 理解 QTextStream 是如何用来支持不同字符集的。