



个人资料

[管理]

一去、二三里

Qing

微博

博客等级: **18**
 博客积分: **78**^新
 博客访问: **407,811**
 关注人气: **378**
 获赠金笔: **12**
 赠出金笔: **0**
 荣誉徽章:

相关博文

- QThread报错:Cannotcreatechildr
筛子喂驴
- Qt之QTcpServer/QTcpSocket简单收
liuhong1.happy
- [转载]基于QTP+QC自动化测试框架
小小
- 解通达信信息文件(gbbq)成功
南上北下
- Qt之线程(QThread)
一去、二三里
- Qt之模型/视图(自定义风格)
一去、二三里
- Qt之QTableView
一去、二三里
- QTableView添加复选框
一去、二三里
- Qt实现网络播放器
一去、二三里
- Qt之QFileIconProvider续(获取文
一去、二三里)

更多>>

正文

字体大小: 大 中 小

开始使用  (2013-12-19 20:42:53) [编辑][删除]

 转载 ▼

标签: qt qml helloworld qtsdk qml基础 分类: QML

本章将介绍使用Qt5开发。我们将告诉你如何安装Qt SDK，如何使用Qt Creator IDE创建以及运行一个简单的Hello World应用程序。

一、安装Qt5 SDK

Qt SDK包括构建桌面或嵌入式应用所需的工具，最新版本可以从Qt-Project homepage上获取（推荐方式）。该SDK本身具有维护工具，可以让你更新SDK到最新版本。

Qt SDK易于安装并配有自己的IDE，促使Qt Creator快速发展。该IDE是一个高效Qt编程环境，推荐给所有的读者。在任何情况下，Qt可以在命令行中使用，也可以使用自己选择的代码编辑器。

安装SDK时，应该选择默认选项，并确保Qt5.x已启用，那么就准备好了。

二、Hello World

为了测试安装，我们将创建一个小型的Hello World应用程序。请打开的Qt Creator并创建一个Qt Quick图形界面项目（File->New File or Project->Qt Quick Project->Qt Quick UI），并将该项目命名HelloWorld。

注：

Qt Creator IDE可以让你创建多种类型的应用程序。如果没有特别声明，我们总是使用Qt Quick UI项目。

Qt Creator中会给你创建几个文件。**HelloWorld.qmlproject**是有关项目配置存储的项目文件。该文件是由Qt Creator管理，所以不要编辑。

另一个文件—**HelloWorld.qml**，是我们的应用程序代码。打开并试着猜测应用程序做了什么。

```
// HelloWorld.qml
import QtQuick 2.0

Rectangle {
    width: 360
    height: 360
    Text {
        anchors.centerIn: parent
        text: "Hello World"
    }
    MouseArea {
        anchors.fill: parent
        onClicked: {
            Qt.quit();
        }
    }
}
```

HelloWorld.qml是用QML语言写的。我们将在下一章对QML语言做深入讨论。只是大多数的描述在一系列分层的元素的用户界面上，特别是这段代码显示一个中心包含“Hello World”文本的360x360像素的矩形。鼠标区域跨越整个矩形，当用户点击时，程序退出。

运行程序，请按左边的▶运行工具或从菜单中选择Build->Run。如果一切顺利的话应该看到这样的界面：

▪ 2550mAh电

▪ 日本雾霾之战：民众与政府的博弈

▪ 南非猎豹冒死捕杀豪猪被扎满嘴刺

▪ 别光顾着把枪口对准柴静

▪ 马来西亚男子与眼镜王蛇接吻(图)

▪ 【早评】降息利好将会在后市中逐

▪ 哪些人能从柴静的“穹顶之下”赚

▪ “深圳机场撞人事件”不是一个人

▪ 政采剔除国外品牌：早该说不了！

▪ 【DIY】做一枚宫灯迎元宵



青蛙冒险坐鳄鱼鼻子



广西北海银滩美景



萨尔瓦多城铁笼监狱



狮子金沙禁猎区旅馆



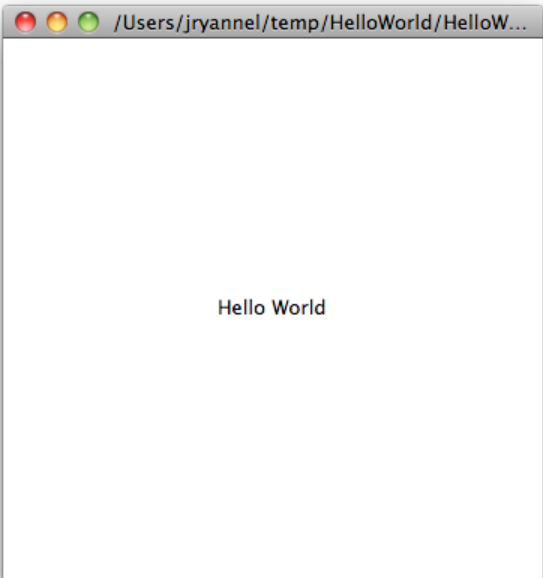
潜水员勇喂巨型双髻鲨



俄罗斯山中隐士生活

查看更多>>

谁看过这篇博文		
	moonlqr	2月26日
	袁伟康1993	2月10日
	墨碧君	2月4日
	xx_xx	2月3日
	鑫语馨缘	1月30日
	Hsby的天涯	1月29日
	码仔救贼	1月20日
	yuanjjff	1月14日
	荧光棒	1月14日
	小灰	1月8日
	乖乖醉长安	1月7日
	夕下	1月5日



三、应用程序类型

这一部分是一个贯穿不同类型的程序类型，有人可能在Qt5用到。它不局限于所提出的选择，但应该给读者一个更好的主意——关于Qt5能够做些什么。

3.1、控制台应用程序

控制台应用程序不提供任何图形人机界面，通常被称作系统服务的一部分或者命令行，或者在命令行中。Qt5也有一系列现成的组件，帮你跨平台的和非常有效的创建控制台应用程序。例如网络API或文件API。还字符串处理或自Qt5.1开始高效的命令行解析器。Qt是一个在C++之上的让你获取编程速度与执行速度的高级API。不要认为Qt只是UI工具包，它提供了很多。

字符串处理

第一个例子，我们演示如何很简单的添加2个常量字符串。这个程序不是非常有用，但让你知道没有一个事件循环的本机C++应用程序看起来是什么样的。

```
// module or class includes
#include

// text stream is text-codec aware
QTextStream cout(stdout, QIODevice::WriteOnly);

int main(int argc, char** argv)
{
    // avoid compiler warnings
    Q_UNUSED(argc)
    Q_UNUSED(argv)
    QString s1("Paris");
    QString s2("London");
    // string concatenation
    QString s = s1 + " " + s2 + "!";
    cout << s << endl;
}
```

容器类

本示例将一个list和list迭代应用程序。Qt提供了大量的容器类，易于使用，并使用相同的API范式。

```
QString s1("Hello");
QString s2("Qt");
QList<QString> list;
// stream into containers
list << s1 << s2;
// Java and STL like iterators
QListIterator<QString> iter(list);
while(iter.hasNext()) {
    cout << iter.next();
    if(iter.hasNext()) {
        cout << " ";
    }
}
```

```

}
cout << "!" << endl;

```

这里，我们将展示一些先进的list功能，允许给一个字符串中加入一系列字符串。当进行基本的文本输入时非常方便。倒转（string->string-list）也可以使用的QString::split()函数。

```

QString s1("Hello");
QString s2("Qt");
// convenient container classes
QStringList list;
list << s1 << s2;
// join strings
QString s = list.join(" ") + "!";
cout << s << endl;

```

文件IO

在接下来的片段，我们从本地目录读取一个CSV文件，并遍历所有的行提取每一行的单元格。这样，我们就可以从ca的CSV文件中得到表数据。20行代码，文件读取给我们提供的只是一个字节流，以便能够把它转换成有效的Unicode文本，我们需要使用的文本流，并通过文件作为一个低级别流。写CSV文件你只需要在写入模式下打开文件，通过管道把那些行插入到文本流。

```

QList<QStringList> data;
// file operations
QFile file("sample.csv");
if(file.open(QIODevice::ReadOnly)) {
    QTextStream stream(&file);
    // loop forever macro
    forever {
        QString line = stream.readLine();
        // test for empty string 'QString("")'
        if(line.isEmpty()) {
            continue;
        }
        // test for null string 'String()'
        if(line.isNull()) {
            break;
        }
        QStringList row;
        // for each loop to iterate over containers
        foreach(const QString& cell, line.split(",")) {
            row.append(cell.trimmed());
        }
        data.append(row);
    }
}
// No cleanup necessary.

```

对于Qt基于控制台的应用程序部分就到这里。

3.2、窗口应用程序

基于控制台的应用程序非常方便，但有时需要有一些图形界面来显示。而且基于图形界面的应用程序将需要一些后台文件的读取/写入，通过网络进行通信或将数据保存在一个容器中。

第一个片段为基础的窗口应用程序，做的较少——创建一个窗口并显示它。没有父亲的窗体在Qt的世界是一个窗口。我们使用的范围指针，以确保作用域指针超出范围时部件删除。应用程序对象封装了Qt运行时，并使用exec调用开始事件循环。从那里上的应用程序重新仅作用于通过鼠标或键盘或类似网络或文件IO等事件提供触发的事件。当事件循环退出该应用程序将退出，这是通过调用‘quit()’来完成对窗口的关闭。

自定义部件

当工作在需要自定义部件（widget）的用户接口上。通常情况下一个小部件是一个充满绘画调用的窗口区域。附加部件有内置的知识关于如何处理键盘或鼠标输入在外部触发反应。要在Qt中做到这一点，我们需要从QWidget派生并重写一些绘制和事件处理的函数。

```

#ifndef CUSTOMWIDGET_H
#define CUSTOMWIDGET_H

#include

class CustomWidget : public QWidget
{
    Q_OBJECT

```

```
public:
    explicit CustomWidget(QWidget *parent = 0);
    void paintEvent(QPaintEvent *event);
    void mousePressEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent *event);
private:
    QPoint m_lastPos;
};

#endif // CUSTOMMWIDGET_H
```

```
#include

int main(int argc, char** argv)
{
    QApplication app(argc, argv);
    QScopedPointer<QWidget> widget(new CustomWidget());
    widget->resize(240, 120);
    widget->show();
    return app.exec();
}
```

实现时，我们画一个小边界小部件和一个小矩形在鼠标最后的位置，这是非常典型的一个低级别的自定义部件。鼠标或键盘事件改变窗口小部件的内部状态，并触发一个绘画更新。Qt提供了大量的现成桌面小部件，所以这种可能性很高，你不需要这么做。

```
#include "customwidget.h"

CustomWidget::CustomWidget(QWidget *parent) :
    QWidget(parent)
{
}

void CustomWidget::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    QRect r1 = rect().adjusted(10, 10, -10, -10);
    painter.setPen(QColor("#33B5E5"));
    painter.drawRect(r1);

    QRect r2(QPoint(0, 0), QSize(40, 40));
    if(m_lastPos.isNull()) {
        r2.moveCenter(r1.center());
    } else {
        r2.moveCenter(m_lastPos);
    }
    painter.fillRect(r2, QColor("#FFB333"));
}

void CustomWidget::mousePressEvent(QMouseEvent *event)
{
    m_lastPos = event->pos();
    update();
}

void CustomWidget::mouseMoveEvent(QMouseEvent *event)
{
    m_lastPos = event->pos();
    update();
}
```

桌面部件

Qt开发者已经做了所有这一切为您提供了一套桌面小部件，这将看到一个原生的不同的系统。你的工作是再安排这些不同的小部件到一个大的部件容器面板中。在Qt中，一个窗口小部件也可以是其他部件的一个容器，这由父子关系来决定。这意味着我们需要使现成的部件如按钮、复选框、单选按钮做为子部件排列布局到另一个部件中。实现这一目标的方法之一如下。

下面是一个所谓的部件容器的头文件。

```
class CustomWidget : public QWidget
{
    Q_OBJECT
public:
    explicit CustomWidget(QWidget *parent = 0);
```

```
private slots:
    void itemClicked(QListWidgetItem* item);
    void updateItem();
private:
    QListWidget *m_widget;
    QLineEdit *m_edit;
    QPushButton *m_button;
};
```

在实现中，我们使用布局以便更好地安排我们的小部件。根据一些大小策略，当容器部件重新调整大小时，重新布局小部件。在这个例子中，我们有一个列表，输入框和按钮垂直排列，以允许编辑城市列表。我们使用信号和槽连接发送和接收对象。

```
CustomWidget::CustomWidget(QWidget *parent) :
    QWidget(parent)
{
    QVBoxLayout *layout = new QVBoxLayout(this);
    m_widget = new QListWidget(this);
    layout->addWidget(m_widget);

    m_edit = new QLineEdit(this);
    layout->addWidget(m_edit);

    m_button = new QPushButton("Quit", this);
    layout->addWidget(m_button);
    setLayout(layout);

    QStringList cities;
    cities << "Paris" << "London" << "Munich";
    foreach(const QString& city, cities) {
        m_widget->addItem(city);
    }

    connect(m_widget, SIGNAL(itemClicked(QListWidgetItem*)), this,
            SLOT(itemClicked(QListWidgetItem*)));
    connect(m_edit, SIGNAL(editingFinished()), this, SLOT(updateItem()));
    connect(m_button, SIGNAL(clicked()), qApp, SLOT(quit()));
}

void CustomWidget::itemClicked(QListWidgetItem *item)
{
    Q_ASSERT(item);
    m_edit->setText(item->text());
}

void CustomWidget::updateItem()
{
    QListWidgetItem* item = m_widget->currentItem();
    if(item) {
        item->setText(m_edit->text());
    }
}
```

图形绘制

一些问题更好的可视化。如果手头的问题看起来像几何形状的物体，Qt graphics view是一个很好的选择。图形视图在一个场景中排列了简单的几何形状，用户可以与这些形状进行交互，或者使用一种算法定位。填充图形视图需要一个图形视图和一个图形场景。场景附加到视图上并填入图形项。这有一个简短的例子。首先声明视图和场景的头文件。

```
class CustomWidgetV2 : public QWidget
{
    Q_OBJECT
public:
    explicit CustomWidgetV2(QWidget *parent = 0);
private:
    QGraphicsView *m_view;
    QGraphicsScene *m_scene;
};
```

实现中，首先场景被附加到视图。该视图是一个小部件，并安排在我们的容器部件中。最后，添加一个小矩形到场景，然后是在视图中呈现。

```
#include "customwidgetv2.h"
```

```
CustomWidget::CustomWidget(QWidget *parent) :
    QWidget(parent)
{
    m_view = new QGraphicsView(this);
    m_scene = new QGraphicsScene(this);
    m_view->setScene(m_scene);

    QVBoxLayout *layout = new QVBoxLayout(this);
    layout->setMargin(0);
    layout->addWidget(m_view);
    setLayout(layout);

    QGraphicsItem* rect1 = m_scene->addRect(0, 0, 40, 40, Qt::NoPen, QColor("#FFBB33"));
    rect1->setFlags(QGraphicsItem::ItemIsFocusable|QGraphicsItem::ItemIsMovable);
}
```

3.3、自适应数据

到现在我们已经涵盖大部分基本数据类型以及如何使用小部件和图形视图。通常在应用程序中，将需要大量的结构化数据，也需要持久存储。这些数据也需要被显示出来。对于这种情况，Qt使用了模型机。一个简单的模型是字符串列表模型，将会充满字符串，然后附加到一个列表视图里。

```
m_view = new QListView(this);
m_model = new QStringListModel(this);
view->setModel(m_model);

QList<QString> cities;
cities << "Munich" << "Paris" << "London";
model->setStringList(cities);
```

存储或检索数据的另一种流行的方式是SQL。Qt提供了SQLite和其他数据库引擎（例如：MySQL、PostgreSQL...）支持。首先，你需要创建数据库使用一个模式，像这样：

```
CREATE TABLE city (name TEXT, country TEXT);
INSERT INTO city value ("Munich", "Germany");
INSERT INTO city value ("Paris", "France");
INSERT INTO city value ("London", "United Kingdom");
```

使用SQL，我们需要将SQL模块添加到.pro文件中。

```
QT += sql
```

然后，我们就可以用c++打开我们的数据库了。首先，我们需要获取一个新的数据库对象指定的数据库引擎，有了这个数据库对象，我们打开数据库。对于SQLite就足以指定数据库文件的路径。Qt提供了一些高级的数据库模型，其中一个表模型，它使用一个表标识符和一个选项where子句来选择数据。所得到的模型可以被附加到一个列表视图的其它模型。

```
QSqlDatabase db = QSqlDatabase::addDatabase("SQLITE");
db.setDatabaseName('cities.db');
if(!db.open()) {
    qFatal("unable to open database");
}

m_model = QSqlTableModel(this);
m_model->setTable("city");
m_model->setHeaderData(0, Qt::Horizontal, "City");
m_model->setHeaderData(1, Qt::Horizontal, "Country");

view->setModel(m_model);
m_model->select();
```

对于更高层次的模型操作Qt提供了一个排序文件代理模型，它允许在基本形式上进行排序和筛选另一个模型。

```
QSortFilterProxyModel* proxy = new QSortFilterProxyModel(this);
proxy->setSourceModel(m_model);
view->setModel(proxy);
view->setSortingEnabled(true);
```

过滤完成基于列的过滤器和一个字符串作为滤波器的参数。

```
proxy->setFilterKeyColumn(0);
proxy->setFilterCaseSensitive(Qt::CaseInsensitive);
```

```
proxy->setFilterFixedString(QString)
```

过滤器的代理模式更强大所以在这里展示，现在足以记住它的存在。

注：

这是另一种关于使用Qt5开发传统应用程序的概述。桌面移动很快，移动设备将成为我们明天的桌面。移动设备有一个不同的用户界面设计。它们比桌面应用程序更简单化。它们做一件事，简单、专注。动画是一个重要的组成部分，用户界面需要感受活力和流畅，传统的Qt技术并不适合这个市场。

接下来：Qt Quick实现救赎。

3.4、Qt Quick应用程序

在现代软件开发中有一个内在冲突，用户界面比我们的后台服务移动得快得多。在传统的技术里在开发所谓的前台具有与后台相同的速度，其结果都是矛盾的，当客户想要在一个项目中更改用户界面，或在项目中有开发用户界面的想法。敏捷项目，需要敏捷方法。

Qt Quick的提供了一个声明环境，其用户界面（前台）可以像HTML一样声明，后台是本地C++代码。这可以从两个世界得到的。

这是一个简单的Qt Quick用户界面，如下：

```
import QtQuick 2.0

Rectangle {
    width: 240; height: 1230
    Rectangle {
        width: 40; height: 40
        anchors.centerIn: parent
        color: '#FFBB33'
    }
}
```

声明的语言被称为QML，需要运行时来运行它。Qt提供了一个标准的运行时叫qmlscene，但也不是那么难以编写一个自定义的运行时间。为此，我们需要一个quick view并设置主要的QML文档作为来源。剩下唯一的事情就是显示用户界面。

```
QQuickView* view = new QQuickView();
QUrl source = QUrl::fromLocalUrl("main.qml");
view->setSource(source);
view.show();
```

回到前面的例子。在一个例子中，我们使用了C++城市模型。如果我们可以在这个模型中声明QML代码将会很棒。

为了确保这点，首先启动编写前端，看看我们如何使用一个可能的城市模型。这种情况下，前端期望一个名为cityModel对象，其可以用在一个列表视图内。

```
import QtQuick 2.0

Rectangle {
    width: 240; height: 120
    ListView {
        width: 180; height: 120
        anchors.centerIn: parent
        model: cityModel
        delegate: Text { text: model.city }
    }
}
```

为了确保cityModel我们主要重用以前的模型并添加上下文属性到根上下文（根上下文是主文档中的其它根元素）

```
m_model = QSqlTableModel(this);
... // some magic code
QHash<int, QByteArray> roles;
roles[Qt::UserRole+1] = "city";
roles[Qt::UserRole+2] = "country";
m_model->setRoleNames(roles);
view->rootContext()->setContextProperty("cityModel", m_model);
```

警告

这并不完全正确，如SQL表模型在列中包含数据，QML模型期望数据为角色。因此，需要有列和角色之间的映射。请参阅QML and QSqlTableModel页面。

总结

我们已经看到了如何安装Qt SDK和如何创建第一个应用程序，介绍了不同类型的应用程序和Qt的概述，展示了Qt提供应用程序的开发的一些功能。我希望你有一个良好的印象，Qt是一个非常丰富的用户界面工具包，并提供了的应用程序开发人员希望的一切。当然，Qt不限制你，仍可以用到其他库或自己扩展Qt。它还富含支持不同的应用模式：控制台，经典的桌面用户界面和触摸用户界面。

注：

技术在于交流、沟通，转载请注明出处并保持作品的完整性。
作者： ☆奋斗ing♥孩子` 原文：http://blog.sina.com.cn/s/blog_a6fb6cc90101h40g.html。



分享：       

阅读(855) | 评论(6) | 收藏(0) | 已有2人转载▼ | 喜欢▼ | 打印 已投稿到： 排行榜

前一篇：相遇Qt5
后一篇：Qt之模型/视图

评论 重要提示：警惕虚假中奖信息 [发评论]

新浪网友

真不错，我会跟着你学的！加油喔！

2013-12-19 23:21 回复(1)

ydp0407

大神能不能把Qt安装配置的教程写详细一点啊~或者单独写个配置教程吧，我捣鼓了一个星期，一直没有找到很好的安装教程~

win8.1 64位系统 Qt creator3.0.1 Qt5.2.1 然后都安装了我就不晓得该怎么弄了，求指导一下新人吧~~

2014-3-19 23:08 回复(1)









lulu

大神的代码编辑器是用的什么配色方案啊？好高大上啊~看着很舒服！

2014-9-28 20:51 回复(1)

发评论

一去、二三里：

☐  分享到微博  ☐ 匿名评论

验证码： 请点击后输入验证码 收听验证码

发评论

以上网友发言只代表其个人观点，不代表新浪网的观点或立场。

< 前一篇
 相遇Qt5

后一篇 >
 Qt之模型/视图

新浪BLOG意见反馈留言板 不良信息反馈 电话：4006900000 提示音后按1键（按当地市话标准计费） 欢迎批评指正
 新浪简介 | About Sina | 广告服务 | 联系我们 | 招聘信息 | 网站律师 | SINA English | 会员注册 | 产品答疑

Copyright © 1996 - 2014 SINA Corporation, All Rights Reserved
 新浪公司 版权所有