QT程序设计——
QT简介

主要内容

- X window系统介绍
- QT简介及其安装
- QT基础
- designer及其表单UI
- 信号和槽

QT-跨平台应用程序框架

- Qt 是一个用于桌面系统和嵌入式开发的跨平台应用程序框架。它包括一个直观的API和一个丰富的类库，以及用于GUI开发和国际化的集成工具，另外它支持Java™和C++开发。Qt让企业无须重新编写源代码，便可以构建运行在不同桌面操作系统和嵌入式设备上的软件应用程序。针对范围广泛的各行各业，包括Google™、Adobe® Lucasfilm® 和 Skype™，通过在 Qt 一个平台上的软件投资而涵盖其它众多平台，Qt可以缩短投入市场的时间并且提高生产效率。

QT体系结构—跨平台

Qt Application Source Code		
Qt API		
Qt/Windows	Qt/X11	Qt Macintosh
QDI	X Windows	Carbon
Windows	Unix/Linux	Mac OS X

X Window

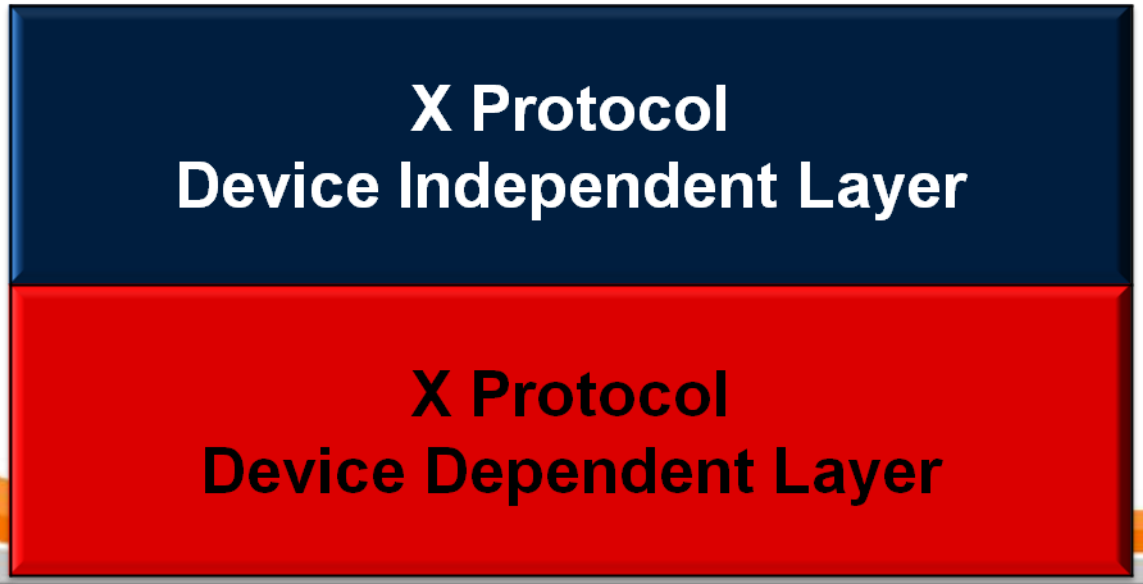
- X Window于1984年在麻省理工学院（MIT）电脑科学研究室开始开发的，当时Bob Scheifler正在发展分步式系统（distributed system），同一时间DEC公司的Jim Gettys 正在麻省理工学院做Athena 计划的一部分。两个计划都需要一个相同的东西－－一套在UNIX机器上运行优良的视窗系统。因此合作关系开始展开，他们从斯坦福（Stanford）大学得到了一套叫做W的实验性视窗系统。因为是根据W视窗系统的基础开始发展的，当发展到了足以和原先系统有明显区别时，他们把这个新系统叫做X。

X window

- X window系统
  - X window系统是建于客户—服务器联结基础上的图形子系统。
  - X window系统独立于系统内核。
- X服务器和客户端
  - X服务器是一个运行在本地计算机上的程序。它响应来自X客户程序的请求，在屏幕上画图或者读取键盘或鼠标的输入，并将它传给客户端。
  - X客户端是一个用诸如Xlib, Xt的库写成的运用X协议的应用程序。它通过向负责管理自己的X服务器提出对显示和输出资源的请求来使用其他计算机的这些资源。

X window

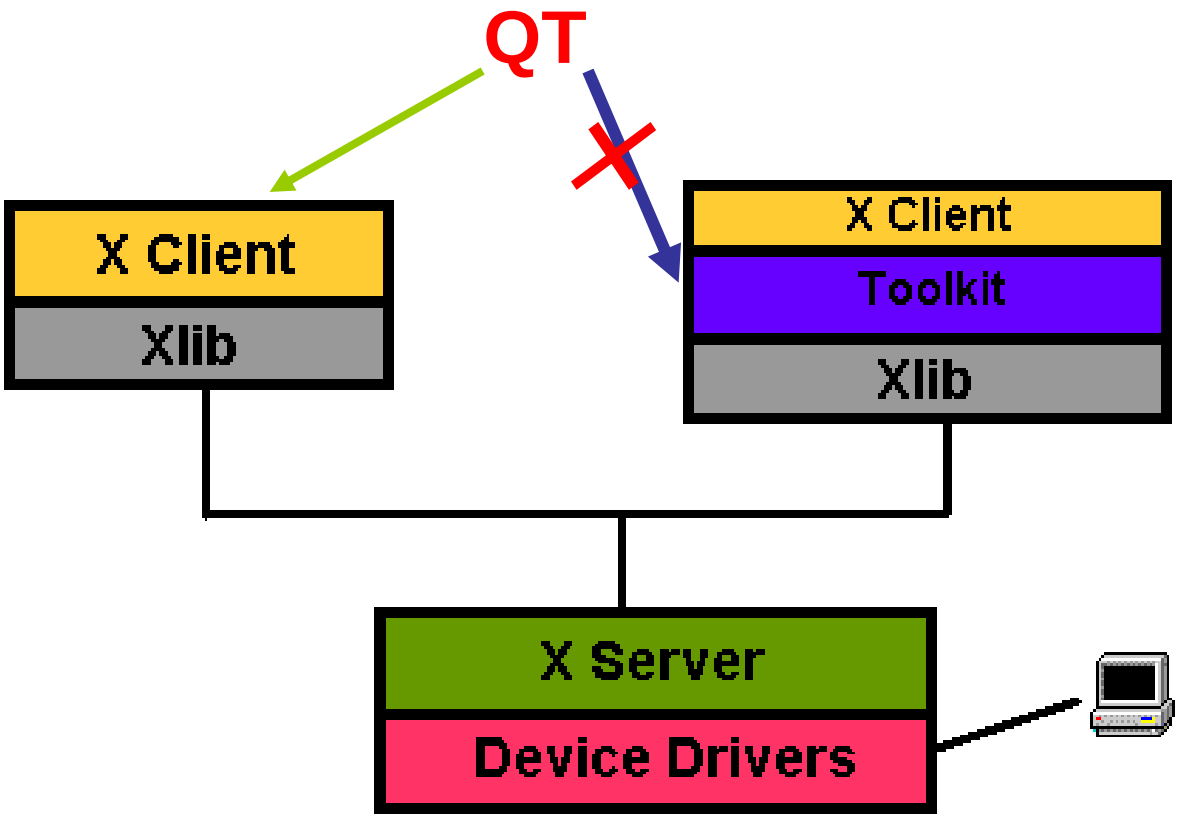
- X协议
  - X协议定义了客户－服务器中应用程序和它的显示的联系。通过这个协议，应用与它的显示被分离开来。



X window

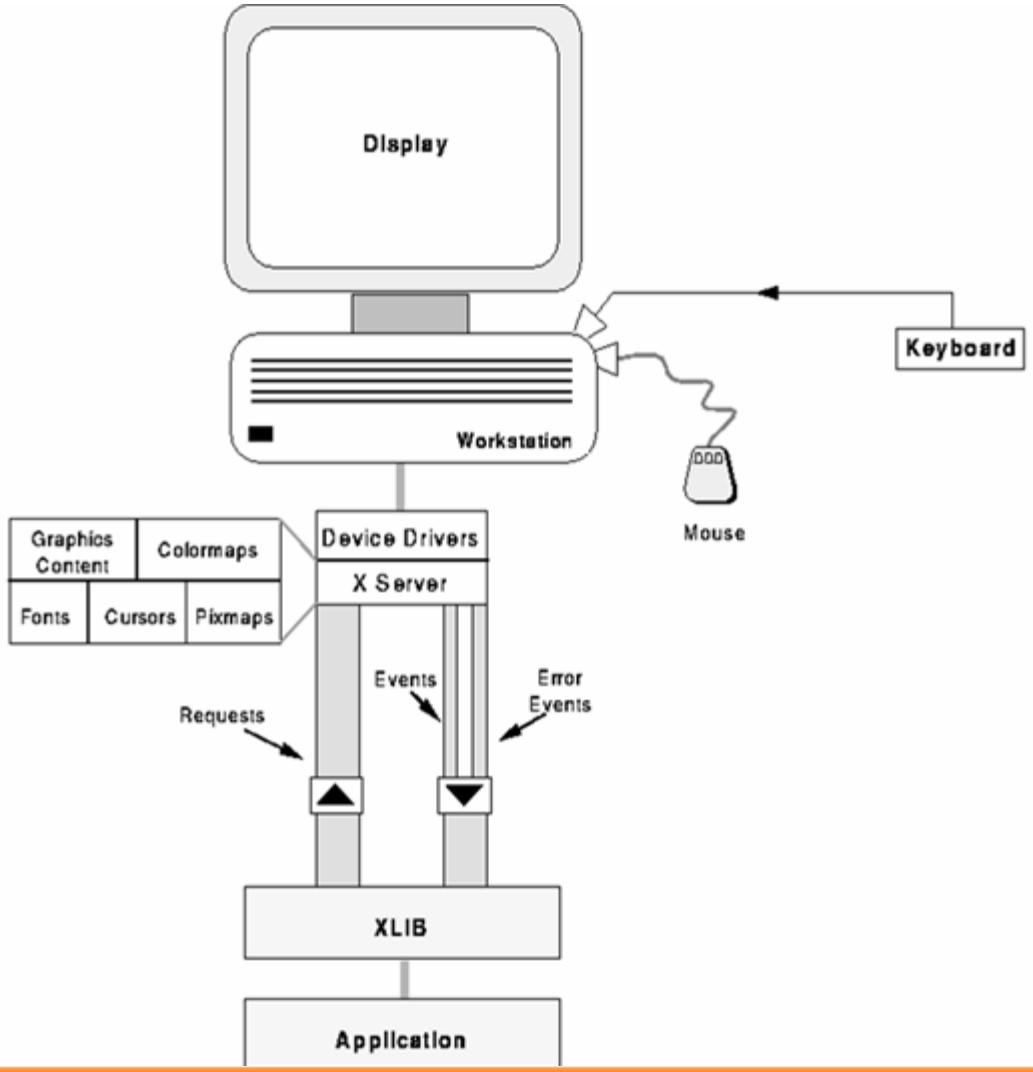
- Xlib库
  - Xlib库是一个C语言库，它为X协议里的信息交换提供了一个API。
- X工具包
  - 将常用的素材（如按钮，菜单等）收集到一起就形成了所谓的X工具包(X ToolKit)
- X窗口管理器
  - X窗口管理器负责安排客户窗口在显示屏幕上的摆放位置，并完成移动窗口和调整窗口尺寸等管理性的工作。

X window


QT
X Client
Xlib
X Client
Toolkit
Xlib
X Server
Device Drivers

X window

这张图清晰的显示了X客户端和X服务器端进行通信的过程。应用程序（X客户端）通过Xlib向X服务器端发出请求，X服务器端接收到请求后，通过设备驱动程序提供的接口，在屏幕上做出相应的显示。X服务器端在接受到鼠标或键盘的动作后，将事件传给x客户端应用程序，使应用程序做出相应的处理。并且，如果X服务器端出现错误，它将会向X客户端发出错误事件的消息。



QT基础

- Trolltech公司(现被Nokia收购)
- 基于C++的GUI开发框架，跨平台
- 技术
  - 丰富的组件
  - 对象的通信机制，信号-槽
  - 方便的事件处理模型
  - 跨平台的GUI应用程序的支持
  - 2D和3D图形支持
  - 国际化支持
  - 数据库
  - 网络编程等
  - 开发工具包

Qt特性

- Qt相对于C++增加的特性有：
  - 能够强有力地支持对象间通信的信号与槽机制
  - 支持可查询和可设计的动态对象属性机制
  - 事件和事件过滤器
  - 基于上下文的字符串本地化
  - 能够支持多任务的定时器
  - 支持按层次检索的对象树
  - 受保护指针
  - 动态类型转换

开发工具包

- 图形设计器:Qt Designer
- 编译工具：qmake
- 本地化工具：Qt Linguist
- 帮助文档:Qt Assistant
- 元对象编译器:moc
- 用户接口编译器:uic
- 资源编译器:rcc

QT/X11安装

- 下载源码http://www.trolltech .com
  qt-x11-opensource-desktop-4.3.4.tar.gz
- 解包解压
  tar xzvf qt-x11-opensource-desktop-4.3.4.tar
- 安装

```
cd 解压目录
./configure
make
sudo make install
```

- 设置环境变量

```
PATH=/usr/local/Trolltech/Qt-4.3.4/bin:$PATH
export PATH
```

QT-” Hello world”

```
#include <QApplication>
#include <QPushButton>
 int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QPushButton hello("Hello world!");
    hello.resize(100, 30);
    hello.show();
    return app.exec();
}
```

QT-”Hello world”

- 编译，运行

```
qmake  -project
qmake
make
**************************
make distclean
```

QT-”Hello world”

- QApplication app(argc, argv);
  - 事件监听循环，派发事件
  - 应用的初始化，应用的析构，提供会话管理
  - 提供系统级的和应用级的配置的设置等
- QPushButton hello(″Hello world!″);
  - 按钮组件

QT简单内建常用组件使用说明

- QLabel 标签
- QButton 按钮
- QLineEdit 单行文本框
- QTextEdit 多行文本框
- QCheckBox 复选框
- QRadioButton 单选框

题目练习

- 把上述”hello world”程序，按钮换成标签组件，单独完成。

QT designer简介

- QT GUI设计器

  1、启动：designer

  2、使用方法：

  演示

  3、项目目录下执行：

  qmake -project 生成项目文件

  qmake 生成Makefile文件

  make 编译（UI工具）

  ****

  UI工具把designer工具生成的表单生成表单类

QT designer简介

- UI工具

```
uic -o HelloForm.h HelloForm.ui
```

- ui表单的使用方法
  1、直接使用
  2、单继承
  3、多继承

QT designer表单应用-1

- UI表单类的直接使用, 如下所示

```
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    QDialog *dag = new QDialog ;
    Ui::Dialog ui;
    ui.setupUi(dag);
    dag->show();

    return app.exec();
}
```

QT designer表单应用-2

■UI表单类的单继承使用,如下所示

```
class HelloForm : public QDialog {
Q_OBJECT
public: HelloForm(QWidget *parent = 0);
private slots:
void on_inputSpinBox1_valueChanged(int value);
void on_inputSpinBox2_valueChanged(int value);
private: Ui::Dialog ui;
};
```

QT designer表单应用-2

- UI表单类的单继承使用,如下所示:

```
HelloForm::HelloForm(QWidget *parent) :
QWidget(parent) {
    ui.setupUi(this);
}
int main(){
    ….
    HelloForm f(0);
    f.show();
    …
}
```

QT designer表单应用-3

- UI表单类的多继承使用,如下所示:

```
class HelloForm : public QDialog, private
   Ui::HelloForm {

   Q_OBJECT
   public: HelloForm(QWidget *parent = 0);

   private slots:
    void on_inputSpinBox1_valueChanged(int value);
    void on_inputSpinBox2_valueChanged(int value);
};
```

QT designer表单应用-3

- UI表单类的多继承使用,如下所示:

```
HelloForm::HelloForm(QWidget *parent) :
    QWidget(parent) {

    setupUi(this);

}
```

信号和槽

- 信号和槽原理
  - 对象之间的通信机制，是QT对C++的扩展
  - 组件中QT预定义的槽和信号

如：QPushButton中预定义信号

- void clicked (bool *checked* = false)
- void pressed ()
- void released ()
- void toggled (bool *checked*)

QPushButton中预定义槽

void setChecked (bool)

等等

信号和槽

- 按钮按下

  按钮按下事件使按钮对象发射（emit） clicked()信号，需要对信号绑定一个槽函数，以便来处理这个信号。

  信号、槽都是类中定义的概念，格式如下

```
class Counter : public QObject {          继承QObject及其子类
    Q_OBJECT                              宏，使用QT元对象提供的功能
    public: Counter() { m_value = 0; }
    int value() const { return m_value; }
    public slots:                         定义槽函数
        void setValue(int value);
    signals: void valueChanged(int newValue);   定义信号
    private: int m_value;
};
```

元对象系统

- 元对象系统(meta-object system)
  - 信号-槽(signal-slot)
  - 内省(introspection)
- 工作机制
  - Q_OBJECT宏声明了在每一个QObject子类中必须实现的一些内省函数：metaObject()、tr()、qt_metacall()……
  - Qt的moc工具生成了用于由Q_OBJECT声明的所有函数和所有信号的实现。
  - 像connect()和disconnect()这样的QObject的成员函数使用这些内省函数来完成它们的工作。

元对象系统特性

- QObject::metaObject() 能够返回与类相关的元对象。
- QMetaObject::classNmae() 能够在不需要实时类型信息（Run-Time Type Information， RTTI）支持的情况下实时返回字符串类型的类名。适合调试。
- QObject::inherits() 能够判断一个类是否继承于另一个类。
- QObject::tr()和QObject::trUtf8()能够为本地化提供字符串翻译。
- QObject::setProperty()和QObject::property() 能够根据名字动态地设置或获取属性。
- QObject::qobject_cast() 能够支持动态类型转换。

Signal和Slot的声明

- 在Qt程序设计中，凡是包含signal和slot的类中都要加上Q_OBJECT的定义，下面的例子给出了如何在一个类中定义signal和slot：

```
class Student : public QObject
{
    Q_OBJECT
    public:
    Student() { myMark = 0; }
    int mark() const { return myMark; }
    public slots:
    void setMark(int newMark);
    signals:
    void markChanged(int newMark);
    private:
    int myMark;
};
```

Signal和Slot的声明

- signal的发出一般在事件的处理函数中，利用emit发出signal，在下面的例子中在在事件处理结束后发出signal

```
void Student::setMark(int newMark)
{
    if (newMark!= myMark) {
    myMark = newMark;
    emit markChanged(myMark);
    }
}
```

Signal和Slot的连接

- 在signal和slot声明以后，需要使用connect()函数将它们连接起来。connect()函数属于QObject类的成员函数，它能够连接signal和slot，也可以用来连接signal和signal
- 函数原形如下：

```
bool connect ( const QObject * sender,
const char * signal,const QObject *
receiver, const char * member ) const
```

- 其中第一个和第三个参数分别指出signal和slot是属于那个对象或组件

Signal和Slot的连接

- 在使用connect()函数进行来接的时候，还需要用到SIGNAL()和SLOT()这两个宏，使用方法如下：

```
QLabel *label = new QLabel;
QScrollBar *scroll = new QScrollBar;
QObject::connect(
scroll,SIGNAL(valueChanged(int)),label,
SLOT(setNum(int)) );
```

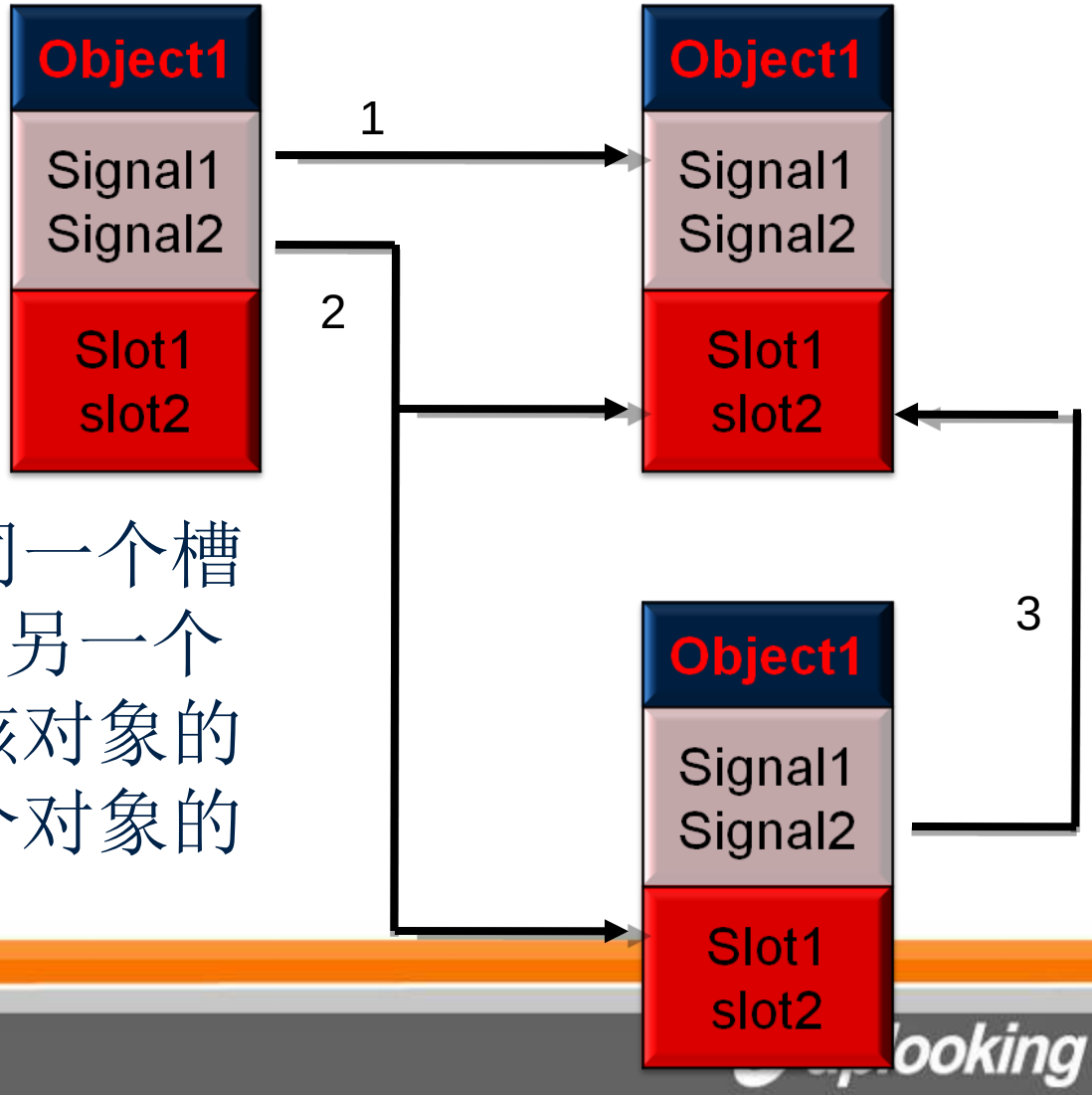
信号和槽

信号和槽的关系：

■一个对象同一信号绑定多个对象的槽函数（包括自身槽）

■多个对象的信号，可以绑定某个对象的同一个槽

■对象的信号可以绑定另一个对象的信号，意思是该对象的信号发射会引起另一个对象的信号发射



信号和槽

- 一个信号可以连接多个槽

```
connect(slider,SIGNAL(valueChanged(int)),
            spinBox,SLOT(setValue(int)));
connect(slider,SIGNAL(valueChanged(int)),
        this,SLOT(updataStatusBar Indicator(int)));
```

当这个信号发射时，槽是一个接一个的调用，而顺序是不确定的。

- 多个信号可以连接一个槽

```
connect(lcd,SIGNAL(overflow()),this,SLOT(handleMathError()));
connect(calculator,SIGNAL(divisionByZero()),
            this,SLOT(handleMathError()));
```

当这个信号发射时，这个槽就会被调用。

- 一个信号可以连接另一个信号

```
connect(lineEdit,SIGNAL(textChanged(const QString
&)),this,SIGNAL(updateRecord(const QString &)));
```

当第一个信号发射时，也会发射第二个信号，除此之外，信号与信号之间的连接和信号与槽之间的连接是难以区分的。

信号和槽

- 连接可以被移除

```
disconnect(lcd,SIGNAL(overflow()),
        this,SLOT(handleMathError()));
```

这种情况比较少用到，因为当删除对象时，Qt会自动移除和这个对象相关的所有连接。

注意：

要把信号成功连接到槽(或者连接到另一个信号)，它们的参数必须具有相同的顺序和相同的类型：

```
connect(ftp,SIGNAL(rawCommandReply(int,const QString
&)),this,SLOT(processReply(int,const QString &)));
```

例外：如果信号的参数比它所连接的槽参数多，那么多余的参数将会被简单地忽略：

```
connect(ftp,SIGNAL(rawCommandReply(int,const QString
&)),this,SLOT(processReply(int)));
```

信号和槽

- 信号和槽

  1、只能在类中声明，不能实现

  2、信号的发射：

  *当某些事件发生时系统自动发射某信号，如点击按钮引起按钮对象发射clicked()信号。

  *当对象状态发生变化，程序员编程发射对象信号

  ```
  emit valueChange(value);
  ```

  3、信号和槽的绑定，信号参数要跟槽的参数匹配，或者槽的参数个数小于信号参数，多余的参数槽会忽略

信号和槽

- 绑定，使用QObect定义的静态connect()函数
- Slots前面的访问修饰符
  - 当把slots当作普通成员函数时，访问修饰符有效
  - 作为槽函数，不起作用。可以连接任何对象的合适的信号。

信号和槽-例子

- 按钮关闭窗体例子

```
QObject::connect(&quit, SIGNAL(clicked()), &app,
    SLOT(quit()));
```

- 例子二

信号和槽限制

- 函数指针无法作为信号与槽的参数

```
class HarbourClass : public QObject
{  Q_OBJECT
public slots:
   void apply(void (*apply)(List *,void *),char *); //error
};
```

解决:

```
typedef void (*ApplyFunction)(List *,void *);
class HarbourClass : public QObject
{  Q_OBJECT
public slots:
   void apply(ApplyFunction,char *);
};
```

信号和槽限制

- 作为信号与槽的参数时，枚举和Typedefs必须完整。

```
class HarbourClass : public QObject
{
    Q_OBJECT
    enmu Error{
        ConnectionRefushed,
        RemoteHostClosed,
        UnknownError
    };
signal:
    void stateChanged(HarbourClass::Error error);
};
```

总结

- X window系统介绍
- QT简介及其安装
- QT基础
- designer及其表单UI
- 信号和槽

作业

■计数器实现

1、实现+ - * / 功能

2、Qstring 类的基本使用方法简介

方法 **toInt**（）

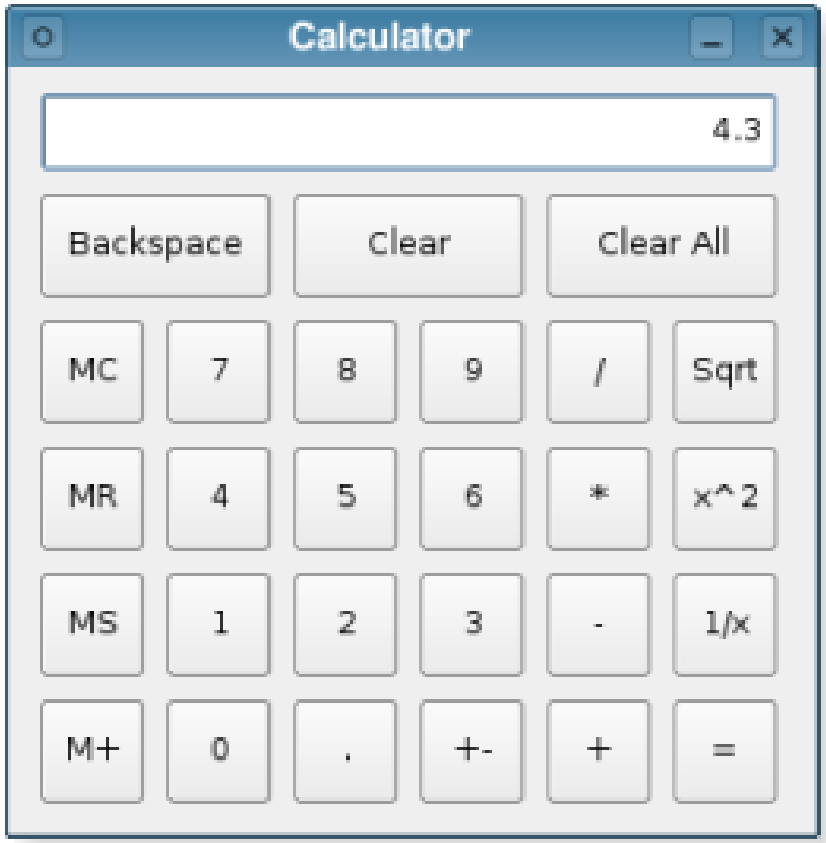
Qstring sq(“7”);

int i=sq.toInt();

3、取得按钮值

button.text()

4、请查阅 相关类的说明，掌握其基本使用方法



后续章节请填写您的邮件地址，订阅我们的精彩内容

嵌入式技术交流QQ群：190018652