# Qt开发基础

- Qt简介
- Qt开发环境
- Qt初步实践
- Qt编程机制
- Qt元对象系统
- QtCreator使用

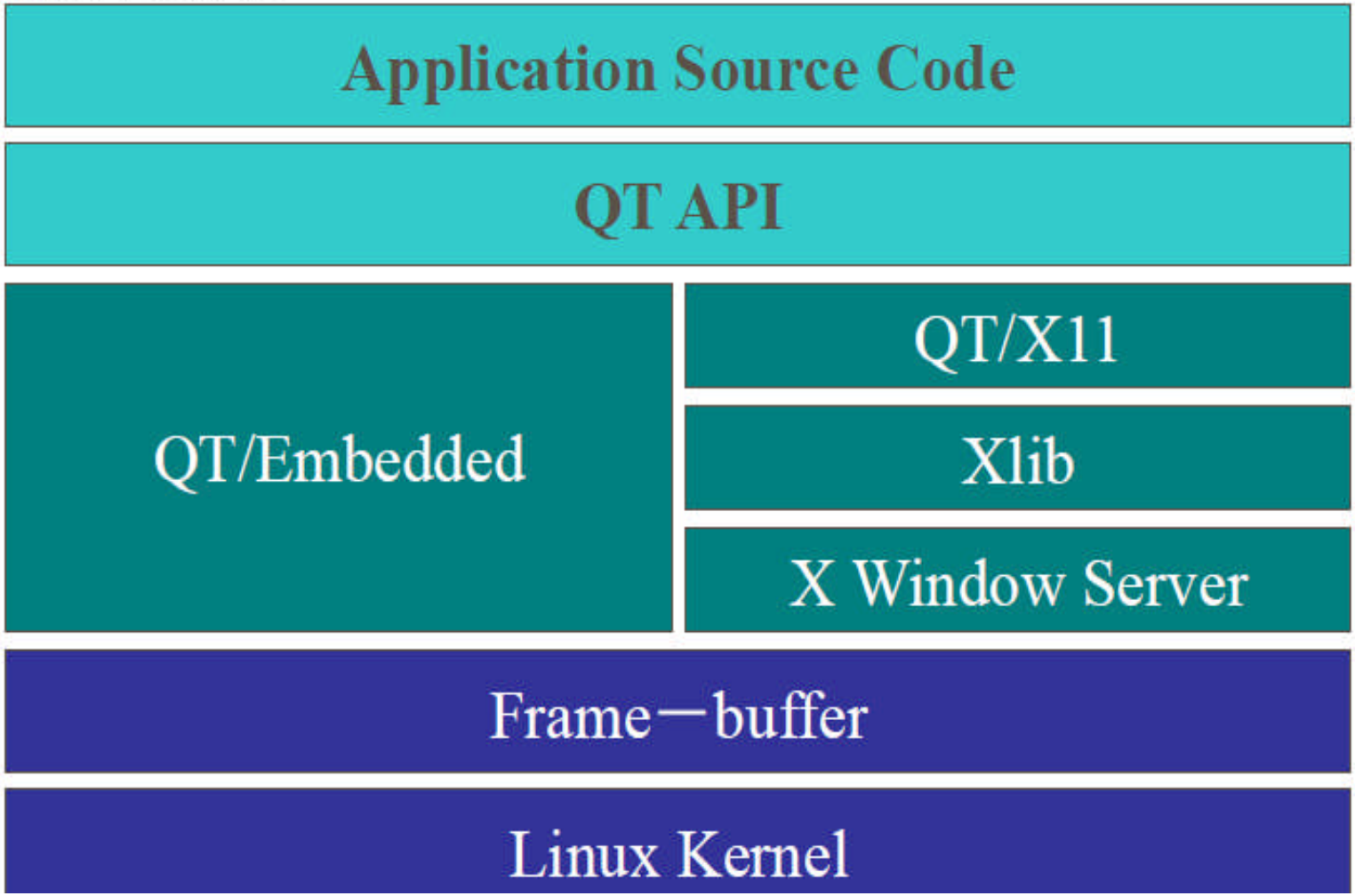
- Qt简介
- Qt是Trolltech（奇趣科技 ）公司的一个产品。
- Qt是一个多平台的C++图形用户界面应用程序框架。它提供给应用程序开发者建立艺术级的图形用户界面所需的所有功能。Qt是完全面向对象的很容易扩展，并且允许真正地组件编程
- Qt支持下述平台：
- MS/Windows -95、98、NT 4.0、ME、和2000
- Unix/X11 -Linux、Sun Solaris、HP-UX、Compaq Tru64 UNIX、IBM AIX、SGI IRIX和其它很多X11平台
- Macintosh -Mac OS X
- Embedded -有帧缓冲(framebuffer)支持的Linux平台。

- **Qt**企业版和**Qt**专业版提供给商业软件开发。它们提供传统商业软件发行版并且提供免费升级和技术支持服务。
- **Qt**自由版是**Qt**仅仅为了开发自由和开放源码软件提供的**Unix/X11**版本。在**GPL**协议下下，它是免费的。
- **Qt/E**自由版是**Qt**为了开发自由软件提供的嵌入式版本。在**GPL**协议下，它是免费的。

# QT的优点

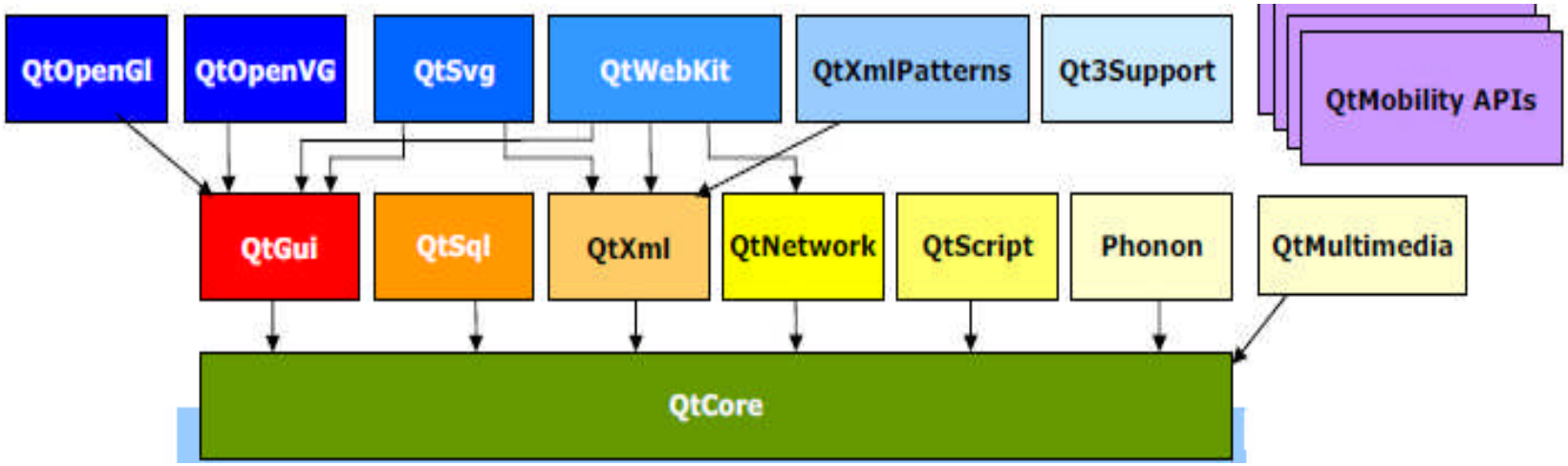
- 可移植性：QT不仅适用于UNIX，同样适用于Windows。为了同时拥有世界上几百万UNIX用户以及几百万的Windows用户，最好的办法是采用一个既适用于UNIX，又适用于Windows的GUI工具包，其答案是QT。
- 易用性：QT是一个C++工具包，它由几百个C++类构成，你在程序中可以使用这些类。因为C++是面向对象的编程语言，而QT是基于C++构造，所以QT具有OOP的所有优点。
- 运行速度：QT非常容易使用，且也具有很快的速度。QT的易用性和快速是密不可分的。这一优点要归功于QT开发者的辛苦工作，他们花费了大量的时间来优化他们的产品。QT比其他许多GUI工具包运行速度快的原因是其实现方式。QT是一个GUI仿真工具包，即它不使用本地工具包作调用，而是使用各自平台上的低级绘图函数，从而提高程序速度。

## ◆QT开发模型


Application Source Code
QT API
QT/Embedded
QT/X11
Xlib
X Window Server
Frame－buffer
Linux Kernel

# Qt 模块

- 跨平台的qt包含了大约15个模块
  - 接近700个APIs，所有的模块都依赖于QtCore
- 编译工具
  - Configure，Qmake，Moc，Uic and Rcc
- 开发工具
  - Qt Creator，Qt Designer，Qt Assistant，Qt Linguist

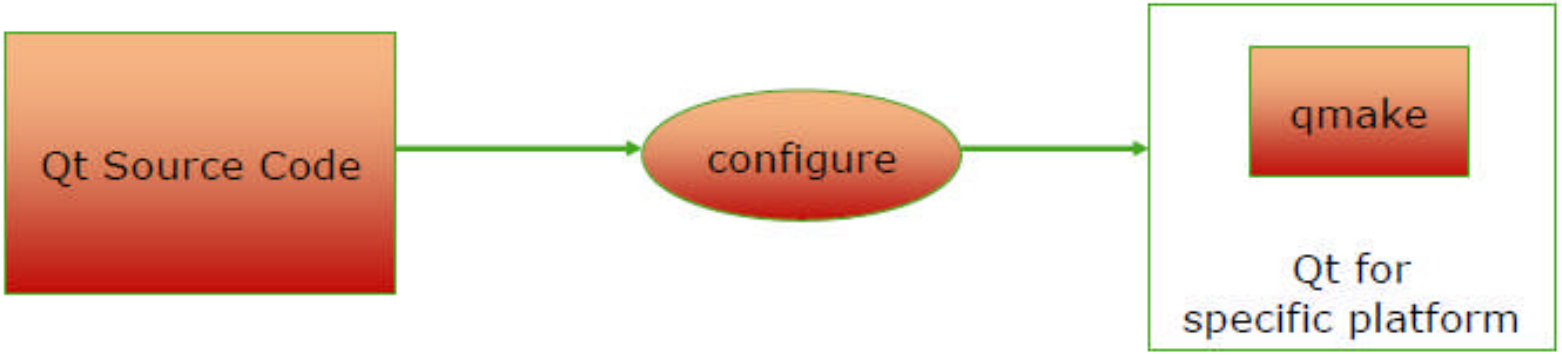


## QT的基本模块

- QtCore，Qt4 的基本模块，定义了其他模块的使用的Qt核心的非GUI类，所有其他的模块都依赖于该模块
- QtGui，定义了图形用户界面类
- QtNetwork，定义了Qt的网络编程类
- QtOpenGL，定义了OpenGL的支持类
- QtSql，定义了访问数据库的类
- QtSvg，定义了显示和生成SVG类
- QtDesigner，定义了在应用程序中直接处理ui文件的类，它使得应用程序能够在运行时使用ui文件构建用户界面
- QtTest，定义了对Qt应用程序和库进行单元测试的类

# 工具– configure

- **configure** 作用是根据当前开发的平台配置**Qt**自身的环境
  - 编译qmake
- 只是在安装**Qt**的时候执行**(**如果**Qt**发布的版本需要手动编译或者是没有当前平台的安装程序**)**



- 工具– moc, uic and rcc

- **moc, Meta-Object 编译器**
  - 对每一个类的头文件，产生一个特殊的meta-object
  - Used by Qt
- **uic, Ui编译器**
  - 根据Qt设计器产生的XML文件生成对应的头文件（包括源代码）
- **rcc, 资源编译器**
  - 生成包含Qt资源文件(.qrc)中数据的C++源文件
- 这些工具在编译的时候自动运行

## Linux 环境设置

```
export PATH=$PATH:$ORACLE_HOME/bin:/home/oracle/qtsdk-2010.05/qt/bin
```

oracle@localhost:/home/oracle

文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

```
[root@localhost oracle]# source .bash_profile
[root@localhost oracle]#
```

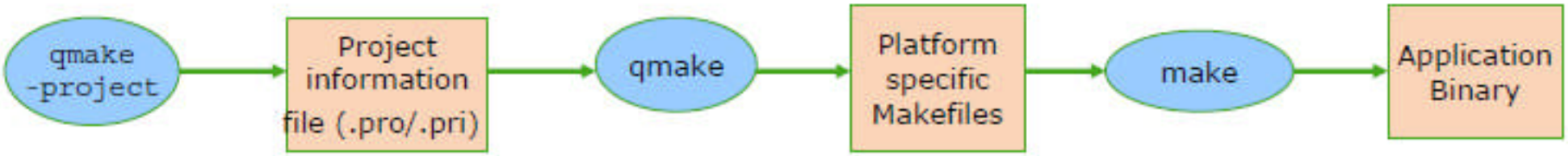
## 一个简单的QT程序

```
#include<QApplication>
#include<QLabel>
int main(int argc, char *argv[ ])
 {
        QApplication app (argc, argv);
        QLabel *hello = new QLabel("Hello  Qt!");
        hello->show();
        return app.exec();
}
```

- 第1 行和第2 行包含了两个头文件，这两个头文件中包含了QApplication 和 QLabel类的定义。
- 第5 行创建了一个QApplication 对象，用于管理整个程序的资源，它需要2 个参数，因为Qt 本身需要一些命令行的参数。
- 第6 行创建了一个用来显示Hello Qt/Embedded!的部件。在Qt 中，部件是一个可视化用户接口，按钮、菜单、滚动条都是部件的实例。部件可以包含其它部件，例如，一个应用程序窗口通常是一个包含QMenuBar、QToolBar、QStatusBar 和其它部件的一个部件。
- 第7 行使hello 部件可视，一般来说部件被创建后都是被隐藏的，因此可以在显示前根据需要来订制部件，这样的好处是可以避免部件创建所造成的闪烁。
- 第8 行把程序的控制权交还给Qt，这时候程序进入就绪模式，可是随时被用户行为激活，例如点击鼠标、敲击键盘等。

# 编译Qt 程序的步骤

- 1. qmake –project
  - 创建**Qt** 工程文件**(.pro).** 工程文件也可以手动创建
- 2. qmake
  - 输入为工程文件，产生平台相关的**Makefile(s)**
  - 产生编译规则，为工程中包含有**Q_OBJECT**宏的头文件调用**moc**编译器（后面介绍**moc**）
- 3. make
  - 编译程序
  - **Executes also** moc**,** uic **and** rcc



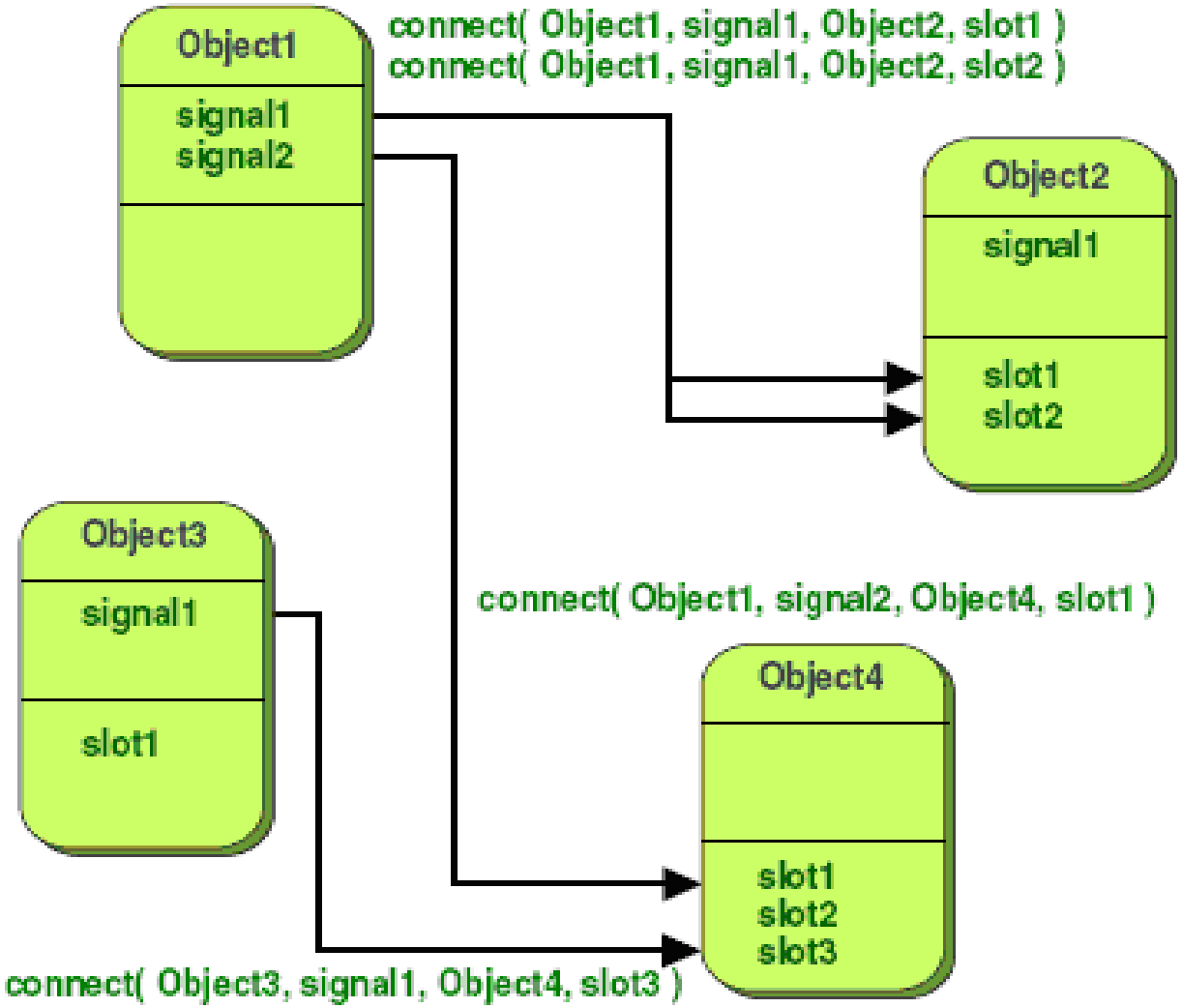
- Qt元对象系统
- 是一个C++扩展，使得QT更适合真正的组件GUI编程
- 使用元编译器moc产生能被标准C++编译器访问的附加C++代码
- 带有moc预编译器的C++基本上提供了面向对象的C的灵活性或类似于Java的运行环境，并保持了C++的执行效率和扩展性
- 实现功能
  - 对象间通信的信号/插槽机制
  - 运行时的类型信息
  - 动态属性系统

- 信号和插槽
- 概述
- 信号与插槽是QT自定义的一种通信机制，它独立于标准的从C/C++语言。他的实现必须借助于moc（Meta Object Compiler）的QT工具，他是一个C++预处理程序，为高层次的事件处理自动生成所需要的附件代码。
- 所谓图形用户接口的应用就是对用户的动作作出响应。程序员则必须把事件和相关代码联系起来，这样才能对事件作出正确的响应。
- 当用户行为发生或者部件状态改变时，**QT**的部件会发射信号。
- 例如，当用户按下按钮时**QPushButton**会发射一个**clicked()**信号。信号可以和一个函数关联起来**(**这个函数称为槽），这样一旦信号发射，槽函数就被自动执行。

## 信号和插槽

- 所有从QObject或其子类（例如Qwidget）派生的类都能够包含信号和插槽。
- 当对象改变状态时，信号就由该对象发射（emit）出来。
- 插槽用于接收信号，但它们是普通的对象成员函数。
- 一个插槽并不知道是否有任何消息与自己相连。用户可以将很多信号与一个插槽相连，也可将单个消息与多个插槽进行链接。

# Signal和Slot的连接方式



## 信号和槽

- 信号和槽通过平滑的扩展**C++**语法并充分利用**C++**的面向对象特性实现。信号和槽是类型安全的，可以重载，也可以重新实现，可以出现在类的公有区、保护区或私有区。
- 若要使用信号和槽，必须继承 QObject 或其子类，并在类的定义中包括 **Q_OBJECT** 宏。信号在类的“信号区”声明，而槽则是在“公有槽区”、“保护槽区”或“私有槽区”中声明的。

- **Signal和Slot的声明（1/2）**

- 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

```
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（2/2）

- **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（1/2）

- 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 QObject::connect (
    const QObject * sender, const char * signal,
        const QObject * receiver, const char * member
  ) [static]
```

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（2/2）**

- 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：

1. QLabel *label = new QLabel;
2. QScrollBar *scroll = new QScrollBar;
3. QObject::connect( scroll,SIGNAL(valueChanged(int)), label,SLOT(setNum(int)) ;

- 取消**Signal**和**Slot**连接

- 取消一个连接
  - disconnect(lcd,SIGNAL(overflow()),this,SLOT(handleMathError()));

- 取消一个连接不是很常用，因为**Qt**会在一个对象被删除后自动取消这个对象所包含的所有的连接

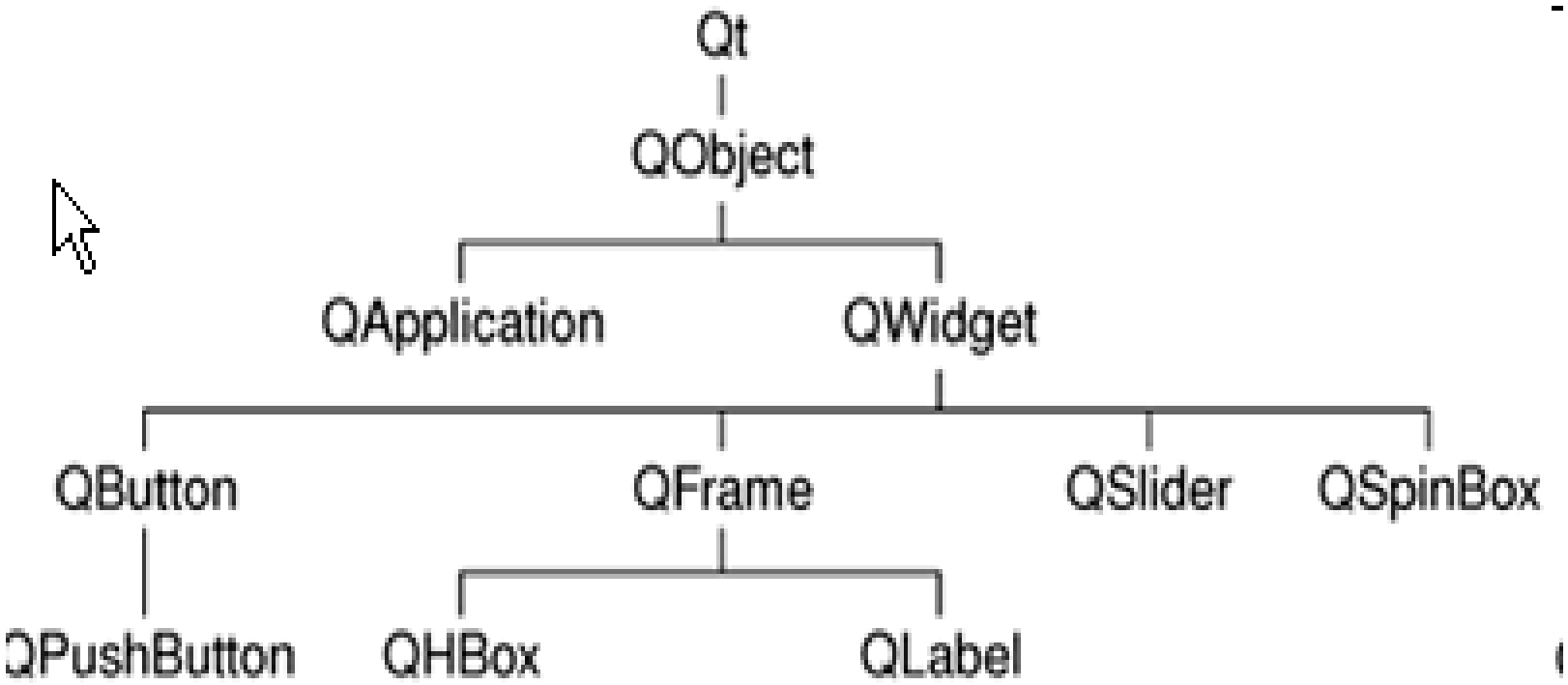
## 信号和槽例子

- 如果“退出”按钮的 **clicked()** 信号与应用程序的 **quit()** 槽相连，那么如果用户单击“退出”，则会终止该应用程序。如果以代码形式表示，则应将上述过程编写为：
- **connect(button, SIGNAL(clicked()), qApp, SLOT(quit()));**

```
1. #include <QApplication>
2. #include <QPushButton>
3. int main(int argc, char *argv[ ])
4. {
5. QApplication app(argc, argv);
6. QPushButton *button = new QPushButton("Quit", 0);
7. QObject::connect(button, SIGNAL(clicked()), &app, SLOT(quit()));
8. button->show();
9. return app.exec();
10. }
```

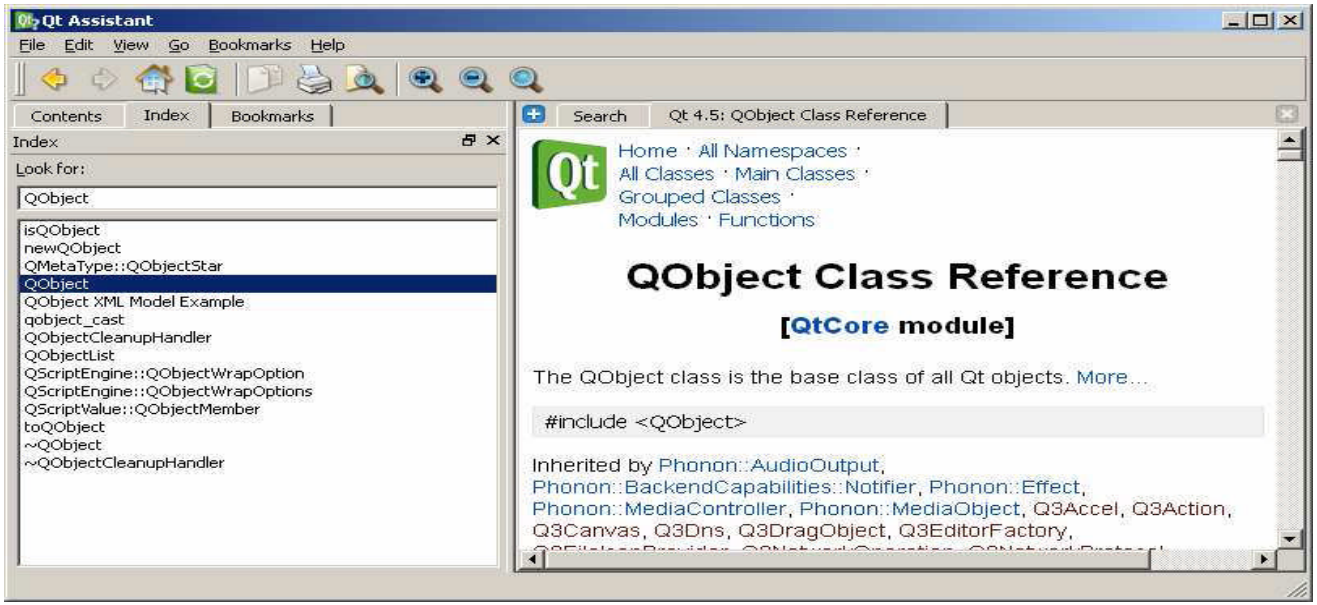
# QT类

- **(Qt 4包含超过700个公共类和逾9000个公共函数接口)**



## Qt 帮助文档

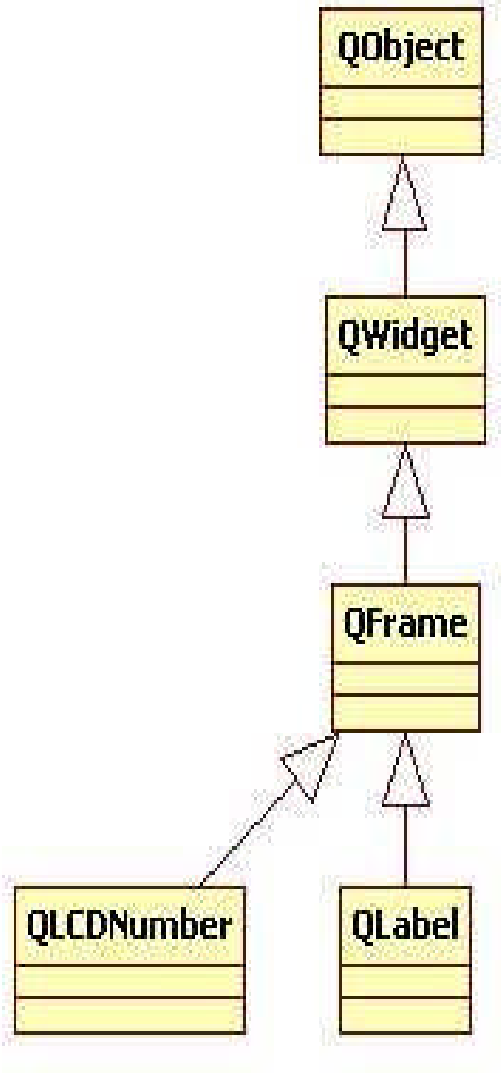
- QT的参考文档对于QT开发者来说是非常重要的工具，因为它涵盖了QT所有的类的函数。
- 我们的程序会用到大量的QT类和函数，但是无法一一讲解，即使讲到也不可能将细节全部展开。因此要深入地了解QT，程序员必须非常熟悉QT参考文档。
- 要运行Qt Assistant, 直接在命令行输入assistant。



- 首页上的**API Reference**链接提供了浏览**QT**类的不同的方法。
- **All Classes**页面列出了**QT**所有的类。
- **Main Classes**页面列出了一些常用的类。
- 在查看应注意一些继承的方法必须在基类中才能查看到。
- 例如，**QPushButton**并没有**show()**这个方法，但是它从基类**QWidget**中继承了这个方法。

# QObject作用

- **Qt对象模型的核心**
  - 是绝大多数类的基类
  - 所有的QWidgets都是QObject
  - 提供对象树和对象的关系
  - QObject在整个Qt的概念体系中处在非常重要的位置。
  - 提供了信号-槽的通信机制
- 具有三个作用
  - 内存管理
  - Introspection（内省）
  - 事件处理



## 父子关系

- 每一个**QObject**实例都可以有一个父亲的参数。
- 孩子会通知他的父亲自己的存在，父亲会把它加入到自己的孩子列表中。
- 如果一个**widget**对象没有父亲，那么他就是一个窗口。
  - 父控件可以:
  - 当父控件隐藏或显示自己的时候，会自动的隐藏和显示子控件。
  - 当父控件Enables and disables时，子控件的状态也随之变化。
  - 注意：在父控件可见的时候，子控件也可以单独隐藏自己。

# 内存管理

- 所有子对象的内存管理都转移给了父对象
  - 使用new在堆上分配内存
  - 子对象可自动被父对象删除内存
  - 手动删除不会引起二次删除，因为子对象删除时会通知父对象
- 没有父对象的QObject对象都需要手动删除
  - 一般把这种无父亲的对象分配在栈上，可以避免内存泄露的问题
- Qt是否有类似于自动回收站的机制？但是事实是没有的！
  - 只需要关注对象的父子关系和功能！

# 创建对象

- 从**QObject**继承的对象都使用**new**在堆上分配空间
  - 如果创建对象的时候设置了父对象，父对象就负责管理新创建的对象
  - QLabel *label = new QLabel(”Some Text”, parent);
- 没有从**Objects**继承的对象则分配在栈上，而不是堆上
  - QStringList list;
  - QColor color;
- 例外
  - QFile and QApplication (inheriting QObject) 通常分配在栈上
  - 模式对话框常常分配在栈上

- **QApplication [QtGui]**
- 管理图形用户界面应用程序的控制流和主要设置。
- 包含主事件循环，来自窗口系统和其它资源的所有事件。
- 处理应用程序的初始化和结束，并且提供对话管理。
- 处理绝大多数系统范围和应用程序范围的设置。
  - 使用用户的桌面设置
    - 色调, 字体
  - 定义程序的外观(look and feel) – setStyle()
  - 提供本地化的字符串
  - Knows application’s windows
    - widgetAt()
  - 利用全局指针qApp访问QApplication实例
  - 继承QCoreApplication [QtCore]
    - 在字符程序中使用(或者是没有任何UI的进程，比如服务器).

# 控件- Widget

- 基类QWidget
  - QWidget类是所有用户界面对象的基类。
  - 窗口部件是用户界面的一个原子：它从窗口系统接收鼠标、键盘和其它事件，并且在屏幕上绘制自己的表现。
  - 每一个窗口部件都是矩形，并且它们按Z轴顺序排列的。一个窗口部件可以被它的父窗口部件或者它前面的窗口部件盖住一部分。
- 其他控件
  - • Label
  - • One line editor
  - • Empty window
  - • Main window
  - • Button
  - • etc.
- • 通常，应用程序都是一个控件，只是这个控件是由很多其它的控件组成

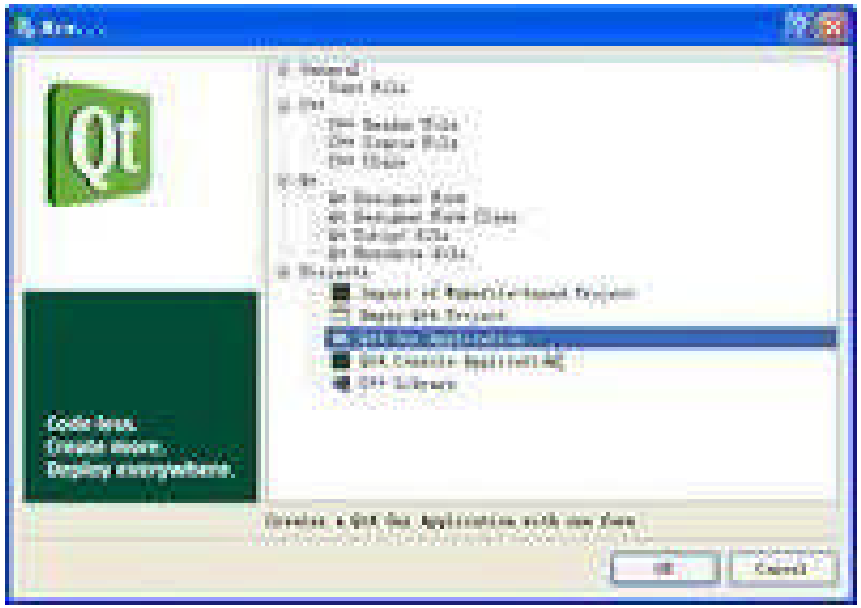
## 窗口与窗口部件

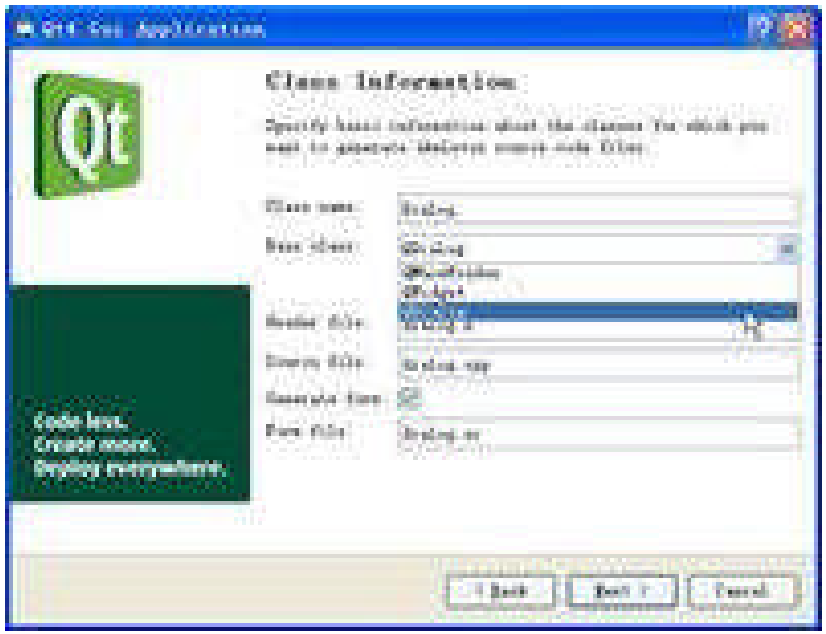
- 窗口
  - 称一个图形界面为窗口，它往往具有标题栏、窗口边框，能够通过鼠标拖动和改变大小属性等特性、最典型的窗口就是对话框。
- 窗口部件
  - 窗口部件是对所有的图形用户界面的统称，它既可以作为单独窗口出现也可以出现在一个窗口的内部。

- **QT建立用户界面的方法很容易理解,也非常灵活.**
  - 1、程序员创建需要的部件并初始化其状态.
  - 2、将部件添加到布局管理器，布局管理器会自动管理部件的尺寸和间距。
  - 3、使用信号和槽机制将部件消息和槽函数关联起来，从而管理用户行为。

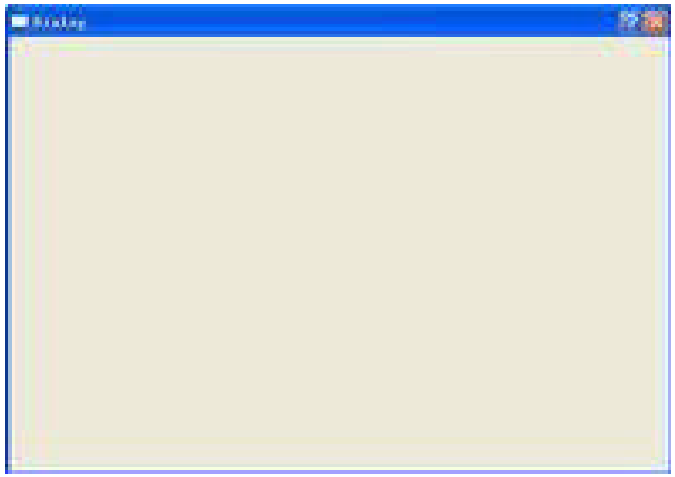
- Qt Creator 的安装和hello world 程序的编写
- 1.首先到Qt 的官方网站上下载Qt Creator，这里我们下载Linux版的。下载完成后，直接安装即可，安装过程中按默认设置即可。
- 2.运行Qt Creator，首先弹出的是欢迎界面，这里可以打开其自带的各种演示程序。

- 3.我们用File->New 菜单来新建工程。
- 4.这里我们选择Qt4 Gui Application。
- 5.下面输入工程名和要保存到的文件夹路径。我们这里的工程名为helloworld。
- 6.这时软件自动添加基本的头文件，因为这个程序我们不需要其他的功能，所以直接点击Next。
- 7.我们将base class 选为QDialog 对话框类。然后点击Next。
- 8.点击Finish，完成工程的建立。

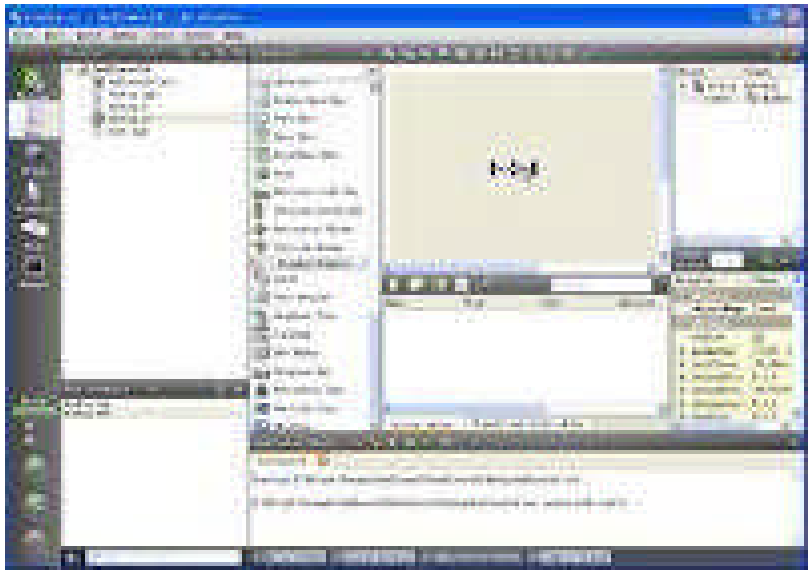




- 9.我们可以看见工程中的所有文件都出现在列表中了。我们可以直接按下下面的绿色的run 按钮或者按下Ctrl+R 快捷键运行程序。
- 注意：
  - QtCreator中要设置qmake编译器路径
  - 如果编译出现undefined reference to `FcFreeTypeQueryFace‘问题
    - 安装fontconfig-2.6.0软件包
      - ./configure --sysconfdir=/etc --prefix=/usr --mandir=/usr/share/man
      - make
      - make install
- 10.程序运行会出现空白的对话框，如下图。



- 11.我们双击文件列表的dialog.ui 文件，便出现了下面所示的图形界面编辑界面。
- 12.我们在右边的器件栏里找到Label 标签器件。
- 13.按着鼠标左键将其拖到设计窗口上，如下图。
- 14.我们双击它，并将其内容改为helloworld。



- 15.我们在右下角的属性栏里将字体大小由9 改为15。
- 16.我们拖动标签一角的蓝点，将全部文字显示出来。
- 17.再次按下运行按钮，便会出现helloworld。

## QtCreator基本工程结构

- 工程文件
- 头文件
- 源代码
- 资源文件
- UI 设计文件

END
Embedded +software+Development
Tony vopo123@126.com