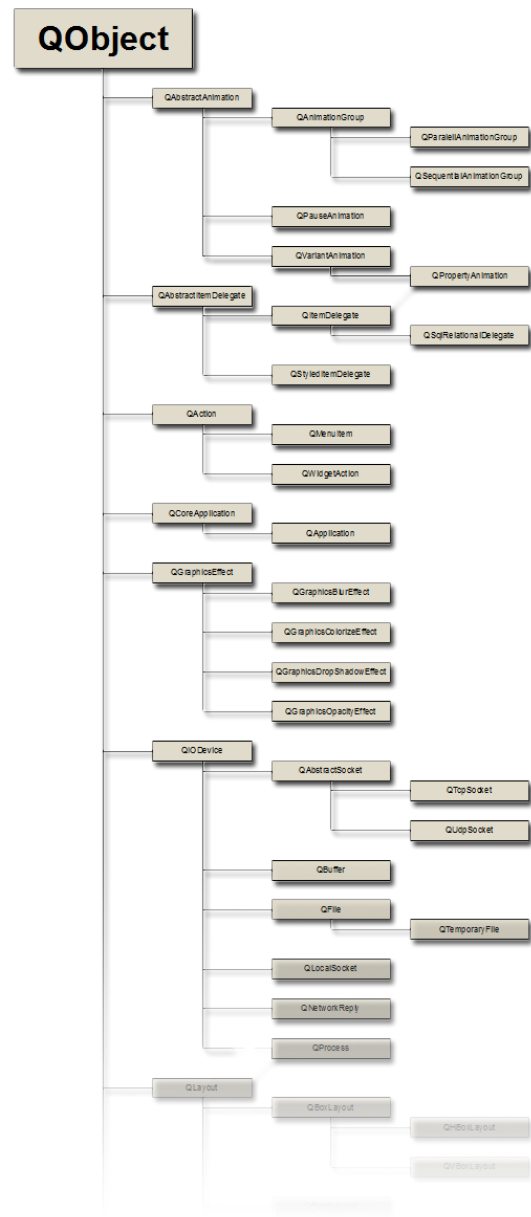


Qt对象模型和信号槽机制

柴强

QObject

- ▶ QObject几乎是所有Qt类和控件的父类
- ▶ 包含构建Qt系统的很多模块
 - 事件机制
 - 信号槽机制
 - 内存管理



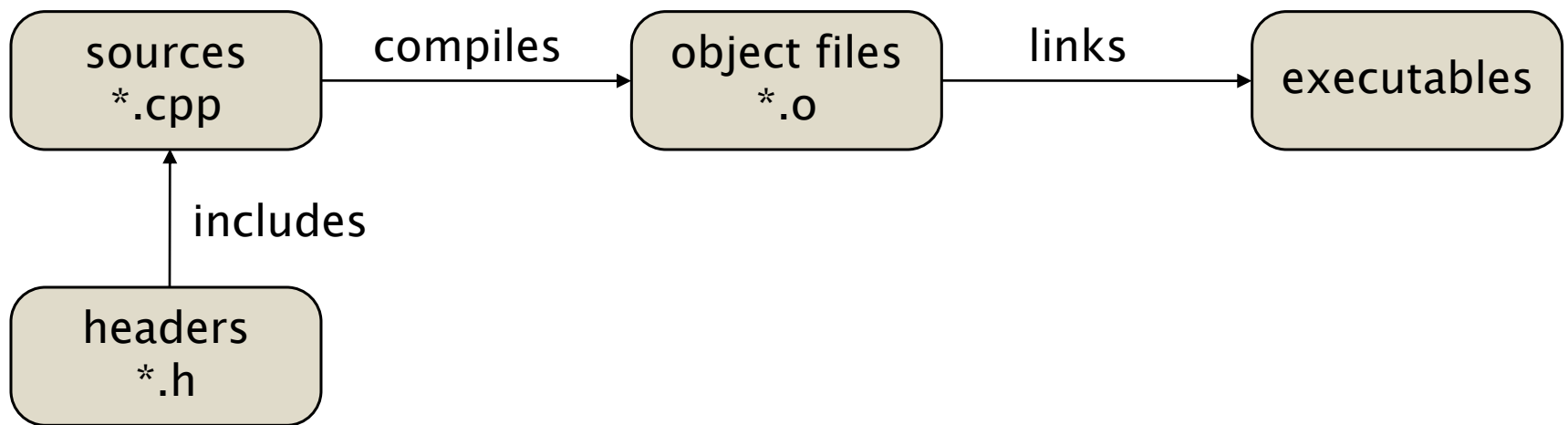
QObject

- ▶ 每个QObject只有一个对象
 - 禁止拷贝构造
- ▶ 元数据
 - 每个QObject有一个元对象
 - 类名className,继承
 - 信号和槽

元数据

- ▶ 元数据是在编译期间由moc编译器生成

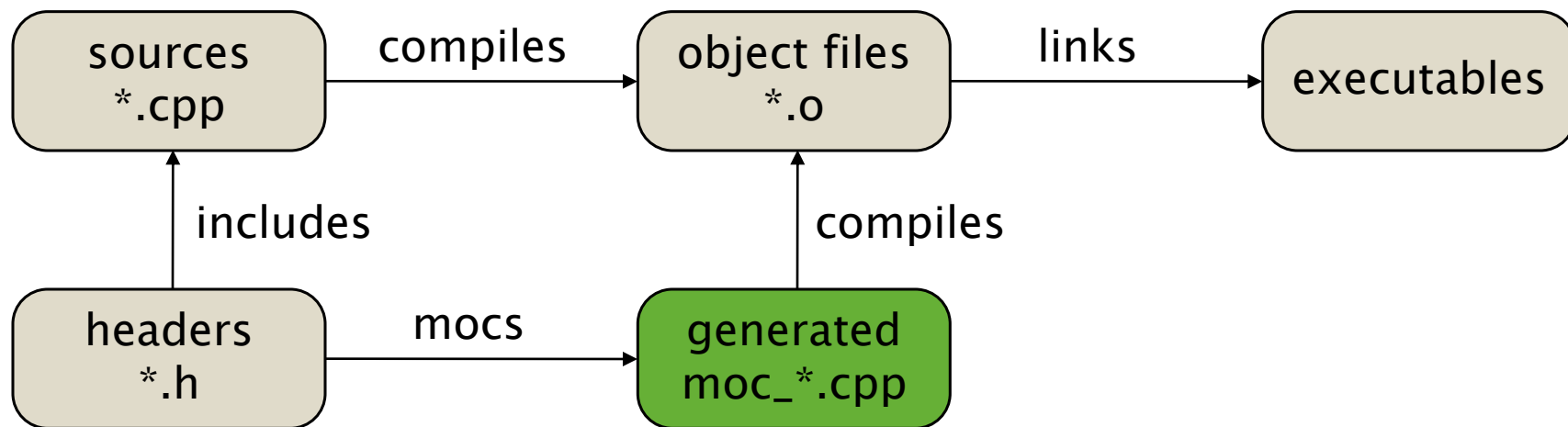
Ordinary C++ Build Process



元数据编译

- ▶ moc从头文件生成元数据

Qt C++ Build Process



MOC

• What does moc look for?

The `Q_OBJECT` macro, usually first

```
class MyClass : public QObject
{
    Q_OBJECT
    Q_CLASSINFO("author", "John Doe")
```

Make sure that you inherit `QObject` first (could be indirect)

General info about the class

```
public:
    MyClass(const Foo &foo, QObject *parent=0);
```

```
    Foo foo() const;
```

```
public slots:
    void setFoo( const Foo &foo );
```

Qt keywords

```
signals:
    void fooChanged( Foo );
```

```
private:
    Foo m_foo;
};
```

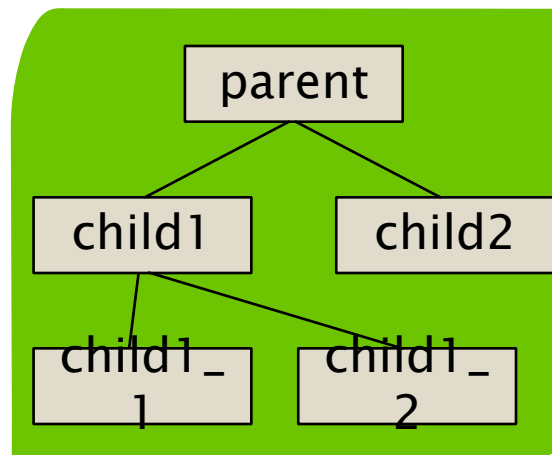
内存管理

- ▶ QObject可以有父亲和孩子
- ▶ 当父对象被删除时，子对象同时被删除

```
QObject *parent = new QObject();  
QObject *child1 = new QObject(parent);  
QObject *child2 = new QObject(parent);  
QObject *child1_1 = new QObject(child1);  
QObject *child1_2 = new QObject(child1);
```

```
delete parent;
```

parent deletes child1 and child2
child1 deletes child1_1 and
child1_2



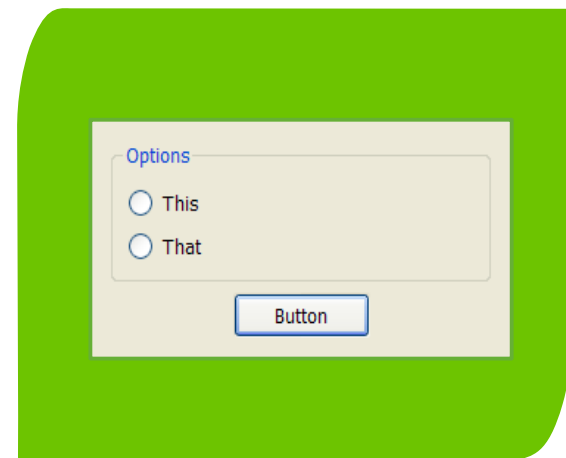
内存管理的使用

► 构建用户界面

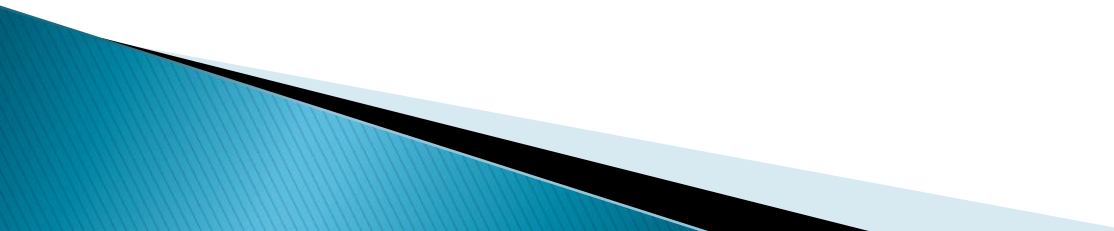
```
QDialog *parent = new QDialog();  
QGroupBox *box = new QGroupBox(parent);  
QPushButton *button = new QPushButton(parent);  
QRadioButton *option1 = new QRadioButton(box);  
QRadioButton *option2 = new QRadioButton(box);
```

```
delete parent;
```

parent deletes box and button
box deletes option1 and
option2



信号槽机制

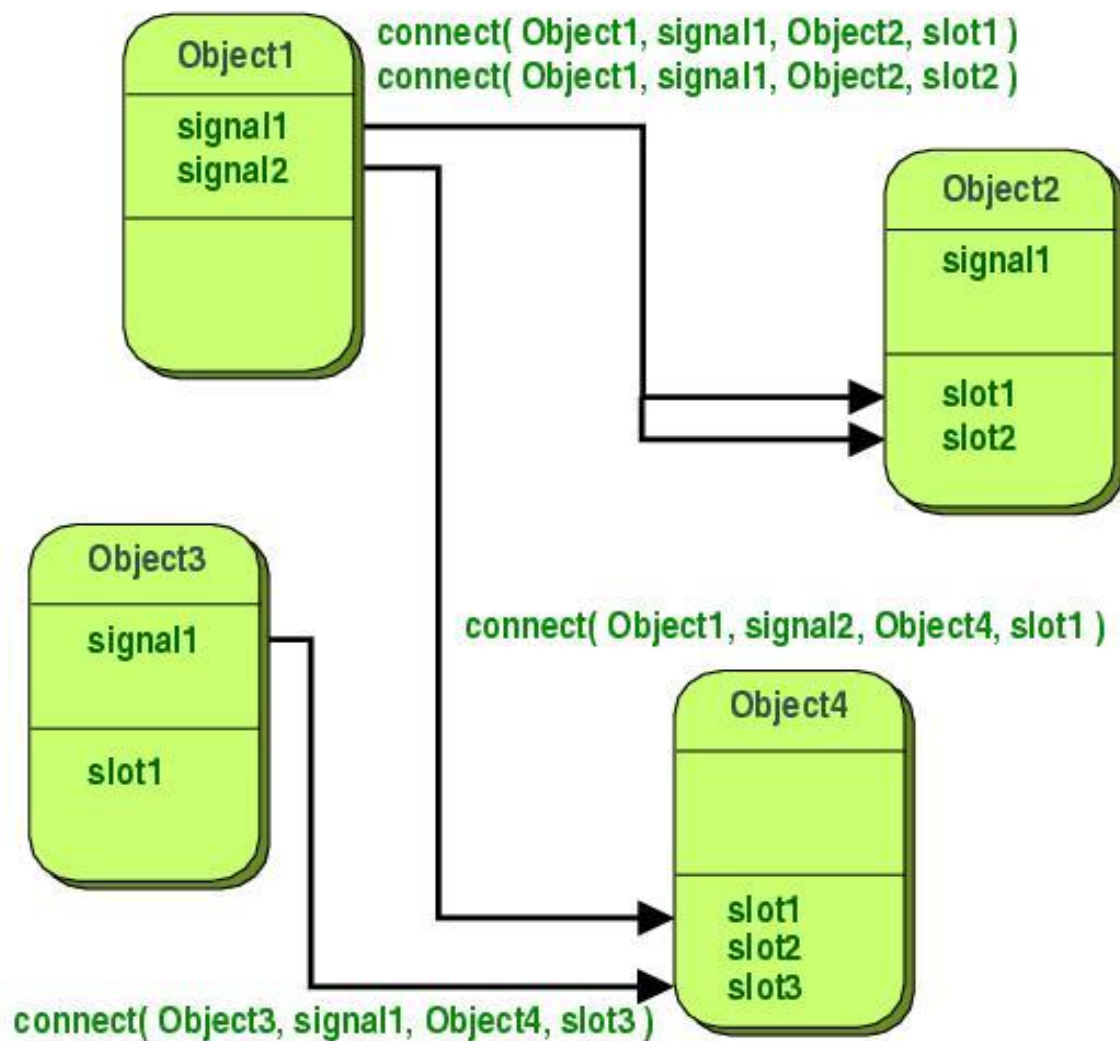
- ▶ Qt的重要机制
 - ▶ 实现事件和界面实现的松耦合
 - ▶ 不同于GTK的Callback模式
 - ▶ 类型安全的回调函数
 - ▶ 多对多的关系
 - ▶ 在QObject中实现
- 

理解信号与槽

▶ 信号与槽功能是Qt特有的

术 语	解释
用户事件	指程序的用户所产生的事件。例如：点击鼠标或拖动滚动条
程序事件	指程序所产生的事件。例如：当用户点击鼠标后程序退出
发射信号	指“发出”或“发送”一个信号。例如：当你点击鼠标时，将发送一个clicked()信号。为了发送这个信号，使用emit关键字。
内部状态	意味着对象内部数据已经发生改变，因此对象发送或发射一个信号
MOC	元对象编辑器。用于构造用户自己的信号和槽。它处理Qt特有的关键字(例如：emit)，创建合法的C++代码。

信号与槽连接图



MVC模式

- ▶ 观察者模式
- ▶ MVC是观察者模式的一个实例，在系统开发架构中重要的地位和意义
 - Model模式—实现 程序的具体功能，包括核心数据结构和逻辑的处理
 - View视图—负责向用户显示处理结果
 - Controller控制器—接收用户的输入，然后调用模型的处理函数进行处理
 - 视图和控制器往往结合在一起，是用户界面的输出部分和输入部分，视图显示结果给用户，控制器相应用户的操作

槽Slot

- ▶ 当一个信号发出时调用的函数
- ▶ 槽函数除了在被连接到信号上之外，和普通的成员函数没有差别
 - 槽函数可以是公有的，保护的，私有的
 - 槽函数还可以是virtual的
 - 信号和槽都可以被继承
- ▶ 槽函数的声明和实现都需要程序员提供

连接

- ▶ 信号发送的结果
 - 如果信号与槽在同一个线程中，相当于直接函数调用
 - 如果信号与槽在不同线程中，会被延迟调用，也就是下一个事件循环的时候调用
- ▶ 信号和槽的参数列表必须相同
 - 只有信号与槽函数中参数的类型是必要的
 - SIGNAL和SLOT将函数名称转换为字符串

信号与槽的连接

- 一个信号可以被连接到多个槽

```
connect(inputBox, SIGNAL(valueChanged(int)), staticBox, SLOT(setValue(int)));  
connect(inputBox, SIGNAL(valueChanged(int)), okButton,  
SLOT(setTittle(int)));
```

- 多个信号可以被连接到一个槽

```
connect(doPlusBox, SIGNAL(overflow()), this, SLOT(mathError()));  
connect(doMinusBox, SIGNAL(lessZero()), this, SLOT(mathError0()));
```

- 一个信号可以被连接到另外一个信号

```
connect(lineEdit, SIGNAL(textChanged()),  
this, SIGNAL(updateRecord()));
```

- 连接可以随时移除

```
disconnect(doPlusBox, SIGNAL(overflow()), this, SLOT(mathError()));
```

使用已有的信号槽

- ▶ QPushButton的信号clicked()
- ▶ QApplication的槽函数quit()
- ▶ 全局变量qApp

信号槽示例

- ▶ ex-signalslots

Q&A