<https://www.cnblogs.com/findumars/p/6350200.html>

本篇说一下Qt对于脚本的支持, 即QtScript模块.

Qt支持的脚本基于ECMAScript脚本语言, 这个东西又是javascript, jscript的基础. 所以, 一般只要学过javascript就基本会写Qt脚本了. 自此开始, Qt脚本现在就叫javascript.  
    不过作为土人, javascript中有一个prototype的概念, 现在才知道. javascript本没有类的概念, 跟不用说是继承之类的了. 但是凭借prototype的特性, 我们可以实现类似C++中类, 以及类继承等一些特性.  
    prototype是个什么概念? 因为这个单词实在表意不清, 导致我花了很多时间来理解这个. 每个javascript对象都有一个指向另一个对象的引用, 这就是它的prototype. 一个对象的prototype定义了这个对象可以进行的操作集. 用C++来类比的话, 这些操作集是一定是成员函数. 看下面的javascript代码:

function Shape(x, y) {

this.x = x;

this.y = y;

}

Shape.prototype.area = function() { return 0; }

function Circle(x, y, radius) {

Shape.call(this, x, y);

this.radius = radius;

}

Circle.prototype = new Shape;

Circle.prototype.area = function() {

return Math.PI \* this.radius \* this.radius;

}

我们把Circle对象的prototype设置成Shape对象, 实际上就是把Shape对象的prototype赋给了Circle对象, 让Circle对象的初始操作集跟Circle对象是一样的. 之后我们又重载了area()函数, 当然我们还可以加入新的函数. 它对应的C++代码如下:

class Shape

{

public:

Shape(double x, double y) {

this->x = x;

this->y = y;

}

virtual double area() const { return 0; }

double x;

double y;

};

class Circle : public Shape

{

public:

Circle(double x, double y, double radius) : Shape(x, y)

{

this->radius = radius;

}

double area() const { return M\_PI \* radius \* radius; }

double radius;

};

所以, 我们看到了, 对于一个javascript对象来说, 它还包括了一个内部的prototype对象. 对于Qt要用C++来实现类似prototype的功能的话, 除了要写一个javascript中的对应类, 还要写这个类对应的prototype类. 这个东西很高级, 也很麻烦, 所以建议看官方文档: http://doc.trolltech.com/4.3/qtscript.html#making-use-of-prototype-based-inheritance

下面我们来说一下一般怎样从Qt的C++代码中调用Qt的script代码. 假设我们要写一个dialog, 上面有一个QPushButton, 一个QLineEdit. 点击QPushButton的时候, 会弹出一个QMessageBox来显示消息.

a) 直接写Qt的C++代码的话, 只要用signal/slot就行了:

void HelloDialog::helloClicked()

{

QString text = textEdit->text();

text = "Hello " + text;

QMessageBox::information(this, "info", text);

}

b) 现在我们要加入javascript 的支持. 要解决的大概有这么一些问题: javascript中怎么拿到QLineEdit里的字符串? javascript中怎么调用QMessage这个Qt的类? 我们还是先来看代码:

QScriptValue constructQMessageBox(QScriptContext \*, QScriptEngine \*engine) {

return engine->newQObject(new QMessageBox());

}

void HelloDialog::helloClicked()

{

QString helloScript = scriptsDir.filePath("xxx.js");

QFile file(helloScript);

if (!file.open(QIODevice::ReadOnly)) {

return;

}

QTextStream in(&file);

in.setCodec("UTF-8");

QString script = in.readAll();

file.close();

// section 1

QScriptEngine engine;

QScriptValue helloDialog = engine.newQObject(this);

engine.globalObject().setProperty("helloDialog", helloDialog);

// section 2

QScriptValue constructor = engine.newFunction(constructQMessageBox);

QScriptValue qsMetaObject = engine.newQMetaObject(&(QMessageBox::staticMetaObject), constructor);

engine.globalObject().setProperty("QMessageBox", qsMetaObject);

QScriptValue result = engine.evaluate(script);

}

我们先把整个javascript文件读进来, 加入一堆设置, 最后调用QScriptEngine::evaluate()函数来执行这段javascript. QScriptEngine这个类就相当于javascript的解释器.  
    javascript里没有类这个概念, 所有的变量都是var类型. 如果要让Qt的C++类在javascript里运行, 那么先要将它包装(wrap)成一个javascript的类型. 代码的section 1部分把this(即当前的dialog)先做了包装, 然后把包装后的对象加入到javascript的运行环境的全局变量中.  
    接着来解决QMessageBox的问题. 由于javascript中没有类, 继而也就是没构造函数这个概念, 但是当我们在javascript中new一个Qt C++对象的时候, 还是需要调用它的构造函数. 代码的section 2部分先把一个C++回调函数(之所以称为回调函数, 是因为要作为QScriptEngine::newFunction()的参数, signature是固定的)包装成一个QScriptValue, 然后把它和QMessageBox的meta-object信息一起包装成一个QScriptValue, 最后依样画葫芦地加入到javascript的运行环境的全局变量中. 这样我们就能在javascript中new出一个QMessageBox了.  
    有一个很重要问题. 就是Qt的meta-object系统和javascript的调用系统是有对应关系的. 在javascript中, 一个var如果是QObject包装而来, 那么这个QObject的所有property(Q\_PROPERTY声明), signal/slot都是可以在javascript中调用的. 还有就是这个QObject的所有child (指的是包含而不是继承关系), 也是可以直接访问的.  
    看一下javascript代码. 其中greeting和text都是属性:

function showMessage(parent, title, text)

{

var messageBox = new QMessageBox;

messageBox.windowTitle = title;

messageBox.text = text;

messageBox.icon = QMessageBox.Information;

return messageBox.exec();

}

return showMessage(helloDialog, "info", helloDialog.greeting + " " + helloDialog.text);

c) 我们实现了用javascript来控制逻辑. GUI的话, Qt也提供了一种可以直接读取\*.ui的方法: QUiLoader::load()函数. 于是我们连GUI也可以不用直接编译到binary里去了. 我们要做的就是用Qt的C++代码搭一个大概的框架, 加载需要的\*.ui, \*.js文件, 在适当的时候调用适当的javascript函数就行了. 而且\*.ui文件对于每个控件都会有一个objectName的属性, 用uic生成代码的话, 这个值就是变量名, 如果用QUiLoader::load()的话, 这个就被赋给了QObject的objectName这个property. 当我们要在一个QWidget的javascript对象里引用它的子控件的时候, 便能直接用这个objectName来引用. 于是\*.ui 和\*.js文件可以说简直配合的天衣无缝那.  
    还是来看代码, Qt的C++代码没什么好说的, 就看javascript代码:

function showMessage(parent, title, text)

{

var messageBox = new QMessageBox;

messageBox.windowTitle = title;

messageBox.text = text;

messageBox.icon = QMessageBox.Information;

return messageBox.exec();

}

function doClick()

{

var text = dialog.textEdit.text;

showMessage(dialog, "info", "Hello " + text);

}

dialog.show();

dialog.helloButton.clicked.connect(doClick);

return 0;

直接访问子控件是不是清爽多了? 呵呵. 代码见这里. 其它请参考官方文档:  
\*) http://doc.trolltech.com/4.3/ecmascript.html  
\*) http://doc.trolltech.com/4.3/qtscript.html

http://ju.outofmemory.cn/entry/150623