本文参考了以下文章：

Qt Internals & Reversing  
【翻译】Qt内部机制及逆向 QT 的信号与槽机制介绍

本文是参考以上文章作出的，但是文章对象是Qt4的，其解析脚本已不适用于Qt5，本人重新分析了Qt5程序的元数据结构，并给出了解析脚本，方便Qt5程序的逆向[破解](https://www.52pojie.cn/)。  
第一次发贴，有任何疑问请回贴，谢谢。  
  
Qt 的信号/槽机制  
Qt 是一个跨平台的C++图形用户界面应用程序框架。它提供给开发者建立图形用户界面所需的功能，广泛用于开发GUI程序，也可用于开发非GUI程序。  
Qt使用信号(Signal)和槽（Slot）机制用于对象间的通信。可以将信号和槽通过QObject对象的connet函数关联起来。我们可以使用emit（Qt定义的语句）发出某个信号，与该信号关联的槽就会接受到信号进行处理。  
下面是一个简单的Qt5代码：

// tsignal.h

#include <QMainWindow>

#include <QObject>

// 必须继承QObject才能使用信号和槽

class TsignalApp:public QMainWindow

 {

public:

    TsignalApp();

    void slotFileNew();

     Q\_OBJECT

     // 信号声明区

     signals:

         // 声明信号 mySignal()

         void mySignal();

         // 声明信号 mySignal(int)

         void mySignal(int x);

         // 声明信号 mySignalParam(int,int)

         void mySignalParam(int x,int y);

     // 槽声明区

     public slots:

         // 声明槽函数 mySlot()

         void mySlot();

         // 声明槽函数 mySlot(int)

         void mySlot(int x);

         // 声明槽函数 mySignalParam (int，int)

         void mySlotParam(int x,int y);

         TsignalApp\* mySlot2();

 };

tsignal.cpp

#include "tsignal.h"

#include <QMessageBox>

TsignalApp::TsignalApp()

{

    // 将信号 mySignal() 与槽 mySlot() 相关联

    connect(this,SIGNAL(mySignal()),SLOT(mySlot()));

    // 将信号 mySignal(int) 与槽 mySlot(int) 相关联

    connect(this,SIGNAL(mySignal(int)),SLOT(mySlot(int)));

    // 将信号 mySignalParam(int,int) 与槽 mySlotParam(int,int) 相关联

    connect(this,SIGNAL(mySignalParam(int,int)),SLOT(mySlotParam(int,int)));

}

// 定义槽函数 mySlot()

void TsignalApp::mySlot()

{

    QMessageBox::about(this,"Tsignal", "This is a signal/slot sample withoutparameter.");

}

// 定义槽函数 mySlot(int)

void TsignalApp::mySlot(int x)

{

    QMessageBox::about(this,"Tsignal", "This is a signal/slot sample with oneparameter.");

}

// 定义槽函数 mySlotParam(int,int)

void TsignalApp::mySlotParam(int x,int y)

{

    char s[256];

    sprintf(s,"x:%d y:%d",x,y);

    QMessageBox::about(this,"Tsignal", s);

}

void TsignalApp::slotFileNew()

{

    // 发射信号 mySignal()

    emit mySignal();

    // 发射信号 mySignal(int)

    emit mySignal(5);

    // 发射信号 mySignalParam(5，100)

    emit mySignalParam(5,100);

}

# main.cpp

#include "tsignal.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

    QApplication a(argc, argv);

    TsignalApp w;

    w.slotFileNew();

    return a.exec();

}

上面的代码编译后运行，会弹出依次弹出，"This is a signal/slot sample withoutparameter."、"This is a signal/slot sample with oneparameter."、"Tsignal x: y:"的窗口。  
从上面的代码可以看出，Qt定义signals、slots、emit等关键字用于简化Qt程序的开发。Qt使用元对象编译器moc（meta object compiler）将这些关键字翻译成正常的C++代码，以便gcc或者vc编译。以上面的代码为例，tsignal.cpp中的qt关键字将被编译成如下代码：

// moc\_tsignal.cpp

#include <QtCore/qbytearray.h>

#include <QtCore/qmetatype.h>

#if !defined(Q\_MOC\_OUTPUT\_REVISION)

#error "The header file 'tsignal.h' doesn't include <QObject>."

#elif Q\_MOC\_OUTPUT\_REVISION != 67

#error "This file was generated using the moc from 5.4.1. It"

#error "cannot be used with the include files from this version of Qt."

#error "(The moc has changed too much.)"

#endif

QT\_BEGIN\_MOC\_NAMESPACE

struct qt\_meta\_stringdata\_TsignalApp\_t {

    QByteArrayData data[10];

    char stringdata[78];

};

#define QT\_MOC\_LITERAL(idx, ofs, len) \

    Q\_STATIC\_BYTE\_ARRAY\_DATA\_HEADER\_INITIALIZER\_WITH\_OFFSET(len, \

    qptrdiff(offsetof(qt\_meta\_stringdata\_TsignalApp\_t, stringdata) + ofs \

        - idx \* sizeof(QByteArrayData)) \

    )

static const qt\_meta\_stringdata\_TsignalApp\_t qt\_meta\_stringdata\_TsignalApp = {

    {

QT\_MOC\_LITERAL(0, 0, 10), // "TsignalApp"

QT\_MOC\_LITERAL(1, 11, 8), // "mySignal"

QT\_MOC\_LITERAL(2, 20, 0), // ""

QT\_MOC\_LITERAL(3, 21, 1), // "x"

QT\_MOC\_LITERAL(4, 23, 13), // "mySignalParam"

QT\_MOC\_LITERAL(5, 37, 1), // "y"

QT\_MOC\_LITERAL(6, 39, 6), // "mySlot"

QT\_MOC\_LITERAL(7, 46, 11), // "mySlotParam"

QT\_MOC\_LITERAL(8, 58, 7), // "mySlot2"

QT\_MOC\_LITERAL(9, 66, 11) // "TsignalApp\*"

    },

    "TsignalApp\0mySignal\0\0x\0mySignalParam\0"

    "y\0mySlot\0mySlotParam\0mySlot2\0TsignalApp\*"

};

#undef QT\_MOC\_LITERAL

static const uint qt\_meta\_data\_TsignalApp[] = {

 // content:

       7,       // revision

       0,       // classname

       0,    0, // classinfo

       7,   14, // methods

       0,    0, // properties

       0,    0, // enums/sets

       0,    0, // constructors

       0,       // flags

       3,       // signalCount

 // signals: name, argc, parameters, tag, flags

       1,    0,   49,    2, 0x06 /\* Public \*/,

       1,    1,   50,    2, 0x06 /\* Public \*/,

       4,    2,   53,    2, 0x06 /\* Public \*/,

 // slots: name, argc, parameters, tag, flags

       6,    0,   58,    2, 0x0a /\* Public \*/,

       6,    1,   59,    2, 0x0a /\* Public \*/,

       7,    2,   62,    2, 0x0a /\* Public \*/,

       8,    0,   67,    2, 0x0a /\* Public \*/,

 // signals: parameters

    QMetaType::Void,

    QMetaType::Void, QMetaType::Int,    3,

    QMetaType::Void, QMetaType::Int, QMetaType::Int,    3,    5,

 // slots: parameters

    QMetaType::Void,

    QMetaType::Void, QMetaType::Int,    3,

    QMetaType::Void, QMetaType::Int, QMetaType::Int,    3,    5,

    0x80000000 | 9,

       0        // eod

};

void TsignalApp::qt\_static\_metacall(QObject \*\_o, QMetaObject::Call \_c, int \_id, void \*\*\_a)

{

    if (\_c == QMetaObject::InvokeMetaMethod) {

        TsignalApp \*\_t = static\_cast<TsignalApp \*>(\_o);

        switch (\_id) {

        case 0: \_t->mySignal(); break;

        case 1: \_t->mySignal((\*reinterpret\_cast< int(\*)>(\_a[1]))); break;

        case 2: \_t->mySignalParam((\*reinterpret\_cast< int(\*)>(\_a[1])),(\*reinterpret\_cast< int(\*)>(\_a[2]))); break;

        case 3: \_t->mySlot(); break;

        case 4: \_t->mySlot((\*reinterpret\_cast< int(\*)>(\_a[1]))); break;

        case 5: \_t->mySlotParam((\*reinterpret\_cast< int(\*)>(\_a[1])),(\*reinterpret\_cast< int(\*)>(\_a[2]))); break;

        case 6: { TsignalApp\* \_r = \_t->mySlot2();

            if (\_a[0]) \*reinterpret\_cast< TsignalApp\*\*>(\_a[0]) = \_r; }  break;

        default: ;

        }

    } else if (\_c == QMetaObject::IndexOfMethod) {

        int \*result = reinterpret\_cast<int \*>(\_a[0]);

        void \*\*func = reinterpret\_cast<void \*\*>(\_a[1]);

        {

            typedef void (TsignalApp::\*\_t)();

            if (\*reinterpret\_cast<\_t \*>(func) == static\_cast<\_t>(&TsignalApp::mySignal)) {

                \*result = 0;

            }

        }

        {

            typedef void (TsignalApp::\*\_t)(int );

            if (\*reinterpret\_cast<\_t \*>(func) == static\_cast<\_t>(&TsignalApp::mySignal)) {

                \*result = 1;

            }

        }

        {

            typedef void (TsignalApp::\*\_t)(int , int );

            if (\*reinterpret\_cast<\_t \*>(func) == static\_cast<\_t>(&TsignalApp::mySignalParam)) {

                \*result = 2;

            }

        }

    }

}

const QMetaObject TsignalApp::staticMetaObject = {

    { &QMainWindow::staticMetaObject, qt\_meta\_stringdata\_TsignalApp.data,

      qt\_meta\_data\_TsignalApp,  qt\_static\_metacall, Q\_NULLPTR, Q\_NULLPTR}

};

const QMetaObject \*TsignalApp::metaObject() const

{

    return QObject::d\_ptr->metaObject ? QObject::d\_ptr->dynamicMetaObject() : &staticMetaObject;

}

void \*TsignalApp::qt\_metacast(const char \*\_clname)

{

    if (!\_clname) return Q\_NULLPTR;

    if (!strcmp(\_clname, qt\_meta\_stringdata\_TsignalApp.stringdata))

        return static\_cast<void\*>(const\_cast< TsignalApp\*>(this));

    return QMainWindow::qt\_metacast(\_clname);

}

int TsignalApp::qt\_metacall(QMetaObject::Call \_c, int \_id, void \*\*\_a)

{

    \_id = QMainWindow::qt\_metacall(\_c, \_id, \_a);

    if (\_id < 0)

        return \_id;

    if (\_c == QMetaObject::InvokeMetaMethod) {

        if (\_id < 7)

            qt\_static\_metacall(this, \_c, \_id, \_a);

        \_id -= 7;

    } else if (\_c == QMetaObject::RegisterMethodArgumentMetaType) {

        if (\_id < 7)

            \*reinterpret\_cast<int\*>(\_a[0]) = -1;

        \_id -= 7;

    }

    return \_id;

}

// SIGNAL 0

void TsignalApp::mySignal()

{

    QMetaObject::activate(this, &staticMetaObject, 0, Q\_NULLPTR);

}

// SIGNAL 1

void TsignalApp::mySignal(int \_t1)

{

    void \*\_a[] = { Q\_NULLPTR, const\_cast<void\*>(reinterpret\_cast<const void\*>(&\_t1)) };

    QMetaObject::activate(this, &staticMetaObject, 1, \_a);

}

// SIGNAL 2

void TsignalApp::mySignalParam(int \_t1, int \_t2)

{

    void \*\_a[] = { Q\_NULLPTR, const\_cast<void\*>(reinterpret\_cast<const void\*>(&\_t1)), const\_cast<void\*>(reinterpret\_cast<const void\*>(&\_t2)) };

    QMetaObject::activate(this, &staticMetaObject, 2, \_a);

}

QT\_END\_MOC\_NAMESPACE

可以看到，mySignal信号被翻译成如下形式

void TsignalApp::mySignal() {

    QMetaObject::activate(this, &staticMetaObject, 0, Q\_NULLPTR);

}

如果信号带参数，那么会多一次类型形转换的过程：

void TsignalApp::mySignalParam(int \_t1, int \_t2) {

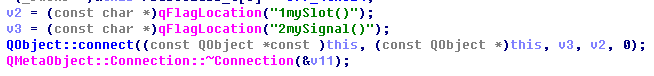
    void \*\_a[] = { Q\_NULLPTR, const\_cast<void\*>(reinterpret\_cast<const void\*>(&\_t1)), const\_cast<void\*>(reinterpret\_cast<const void\*>(&\_t2)) };

    QMetaObject::activate(this, &staticMetaObject, 2, \_a);

}

无论带不带参数，发出信号都会调用的QMetaObject::activate函数。  
  
Qt 元数据结构  
QMetaObject::activate使得Qt可以动态调用信号关联的槽，但这给也我们破解Qt程序带来了困难。  
假设有这样一个程序，在接收到[注册码](https://www.52pojie.cn/)调用后进行检查，如果正确，则触发触发注册成功的信号，否则触发注册失败信号。注册码破解的直接思路是通过注册失败的信息查找注册函数，但是由上面我们可以知道发送信号的时候是通过QMetaObject::activate触发信号，这不是一个直接的函数调用。即使我们在出错信息上下断点，此时栈上是大量的Qt核心库调用信息，触发信号的函数查找起来十分麻烦。  
Qt是通过connect将信号与槽关联。但是connet的时候，我们无法直接知道与信号相关联的槽函数的位置。例如：

connect(this,SIGNAL(mySignal()),SLOT(mySlot()));

在编译后使用[IDA](https://www.52pojie.cn/thread-675251-1-1.html)反编译如下：  
  
  
没有出现mySlot的地址，保留的只有mySlot的名称。  
但是Qt是可以正确调用相应的函数的，原因是Qt会将信号和槽的元数据保存，并在运行的时候动态查找相应的方法。在moc\_tsignal.cpp中，元数据如下：

static const qt\_meta\_stringdata\_TsignalApp\_t qt\_meta\_stringdata\_TsignalApp = {

    {

QT\_MOC\_LITERAL(0, 0, 10), // "TsignalApp"

QT\_MOC\_LITERAL(1, 11, 8), // "mySignal"

QT\_MOC\_LITERAL(2, 20, 0), // ""

QT\_MOC\_LITERAL(3, 21, 1), // "x"

QT\_MOC\_LITERAL(4, 23, 13), // "mySignalParam"

QT\_MOC\_LITERAL(5, 37, 1), // "y"

QT\_MOC\_LITERAL(6, 39, 6), // "mySlot"

QT\_MOC\_LITERAL(7, 46, 11), // "mySlotParam"

QT\_MOC\_LITERAL(8, 58, 7), // "mySlot2"

QT\_MOC\_LITERAL(9, 66, 11) // "TsignalApp\*"

    },

    "TsignalApp\0mySignal\0\0x\0mySignalParam\0"

    "y\0mySlot\0mySlotParam\0mySlot2\0TsignalApp\*"

};

#undef QT\_MOC\_LITERAL

static const uint qt\_meta\_data\_TsignalApp[] = {

 // content:

       7,       // revision

       0,       // classname

       0,    0, // classinfo

       7,   14, // methods

       0,    0, // properties

       0,    0, // enums/sets

       0,    0, // constructors

       0,       // flags

       3,       // signalCount

 // signals: name, argc, parameters, tag, flags

       1,    0,   49,    2, 0x06 /\* Public \*/,

       1,    1,   50,    2, 0x06 /\* Public \*/,

       4,    2,   53,    2, 0x06 /\* Public \*/,

 // slots: name, argc, parameters, tag, flags

       6,    0,   58,    2, 0x0a /\* Public \*/,

       6,    1,   59,    2, 0x0a /\* Public \*/,

       7,    2,   62,    2, 0x0a /\* Public \*/,

       8,    0,   67,    2, 0x0a /\* Public \*/,

 // signals: parameters

    QMetaType::Void,

    QMetaType::Void, QMetaType::Int,    3,

    QMetaType::Void, QMetaType::Int, QMetaType::Int,    3,    5,

 // slots: parameters

    QMetaType::Void,

    QMetaType::Void, QMetaType::Int,    3,

    QMetaType::Void, QMetaType::Int, QMetaType::Int,    3,    5,

    0x80000000 | 9,

       0        // eod

};

此外，每个Qt对象都有一个qt\_static\_metacall方法用于确定调用的函数：

void TsignalApp::qt\_static\_metacall(QObject \*\_o, QMetaObject::Call \_c, int \_id, void \*\*\_a)

{

    if (\_c == QMetaObject::InvokeMetaMethod) {

        TsignalApp \*\_t = static\_cast<TsignalApp \*>(\_o);

        switch (\_id) {

        case 0: \_t->mySignal(); break;

        case 1: \_t->mySignal((\*reinterpret\_cast< int(\*)>(\_a[1]))); break;

        case 2: \_t->mySignalParam((\*reinterpret\_cast< int(\*)>(\_a[1])),(\*reinterpret\_cast< int(\*)>(\_a[2]))); break;

        case 3: \_t->mySlot(); break;

        case 4: \_t->mySlot((\*reinterpret\_cast< int(\*)>(\_a[1]))); break;

        case 5: \_t->mySlotParam((\*reinterpret\_cast< int(\*)>(\_a[1])),(\*reinterpret\_cast< int(\*)>(\_a[2]))); break;

        case 6: { TsignalApp\* \_r = \_t->mySlot2();

            if (\_a[0]) \*reinterpret\_cast< TsignalApp\*\*>(\_a[0]) = \_r; }  break;

        default: ;

        }

    } else if (\_c == QMetaObject::IndexOfMethod) {..

    }

}  
  
其中的\_id是相应Qt方法（信号/槽等）的索引，将索引与方法对应十分简单，之后会说。  
实际上，在编译后Qt程序中qt\_meta\_stringdata\_TsignalApp和qt\_meta\_data\_TsignalApp会被封装在另一个数据结构中，由Qt的源代码分析得，该结构为QMetaObject内部的结构体d：

// qobjectdefs.h

struct QMetaObject {

    struct { // private data

    const QMetaObject \*superdata;

    const QByteArrayData \*stringdata;

    const uint \*data;

    typedef void (\*StaticMetacallFunction)(QObject \*, QMetaObject::Call, int, void \*\*);

    StaticMetacallFunction static\_metacall;

    const QMetaObject \* const \*relatedMetaObjects;

    void \*extradata; //reserved for future use

    } d;

};

在结构体由索引（index）得到对应的方法代码如下：

// qmetaobject.cpp

QMetaMethod QMetaObject::method(int index) const

{

    int i = index;

    i -= methodOffset();

    if (i < 0 && d.superdata)

        return d.superdata->method(index);

    QMetaMethod result;

    if (i >= 0 && i < priv(d.data)->methodCount) {

        result.mobj = this;

        result.handle = priv(d.data)->methodData + 5\*i;

    }

    return result;

}

其中priv(d.data)的代码如下：

static inline const QMetaObjectPrivate \*priv(const uint\* data)

{ return reinterpret\_cast<const QMetaObjectPrivate\*>(data); }

即d.data指向一个QMetaObjectPrivate对象。  
QMetaObjectPrivate的声明如下：

// qmetaobject\_p

struct QMetaObjectPrivate {

    enum { OutputRevision = 7 }; // Used by moc, qmetaobjectbuilder and qdbus

    int revision;

    int className;

    int classInfoCount, classInfoData;

    int methodCount, methodData;

    int propertyCount, propertyData;

    int enumeratorCount, enumeratorData;

    int constructorCount, constructorData; //since revision 2

    int flags; //since revision 3

    int signalCount; //since revision 4

}

实际上，这就是moc\_tsignal.cpp中的相应内容

// content:

      7,       // revision

      0,       // classname

      0,    0, // classinfo

      7,   14, // methods

      0,    0, // properties

      0,    0, // enums/sets

      0,    0, // constructors

      0,       // flags

      3,       // signalCount

由QMetaObjectPrivate.methodData成员，可以确定QMetaMethod在d.data中的偏移，不考虑父类的情况下，索引为id的QMetaMethod在d.data的偏移为QMetaObjectPrivate.methodData + id \* 5  
  
QMetaMethod  
QMetaMethod在Qt源代码中只保留QObject的指针及偏移量计算的代码，没有定义真正的结构体，为了方便，将结构体整理如下：

struct QMetaMethod {

    int name;

    int parameterCount;  // 参数个数

    int typesDataIndex;

    int tag;

    int flag;

}

获得方法名  
  
QMetaMethod其中的name和typesDataIndex都是其在d.stringdata[]的索引。在moc中，由d.stringdata由如下宏生成：

QT\_MOC\_LITERAL(0, 0, 10), // "TsignalApp"

在宏之后紧跟着C语言的字符串数据：

"TsignalApp\0mySignal\0\0x\0mySignalParam\0"

"y\0mySlot\0mySlotParam\0mySlot2\0TsignalApp\*"

d.stringdata指向的类型为QByteArrayData，其定义为：

typedef QArrayData QByteArrayData;

// qarraydata.h

struct QTypedArrayData

    : QArrayData

struct QArrayData

{

    QtPrivate::RefCount ref;

    int size;

    uint alloc : 31;

    uint capacityReserved : 1;

    qptrdiff offset; // 相对于QArrayData所在地址的偏移

};

由QArrayData.offset，我们可以计算C字符串的偏移，由QArrayData.size可以确定字符串的长度。  
至此，我们可以得到QMetaMethod的方法名。  
例如：设mth为QMetaMethod对象，通过d.stringdata[mth.name]获得QArrayData对象arr，则&arr + arr.offset处即为mth的方法名字符串。  
  
获得方法类型数据  
QMetaMethod.typesDataIndex是方法的类型数据偏移，其计算方式如下：  
偏移=d.data + QMetaMethod.typesDataIndex \* 4  
知道类型数据偏移后，我们就可以得到QMetaMethod的类型，以上面例子的mySignalParam为例，其原型为：

`void mySlotParam(int x,int y);

其编译后的类型信息为：

QMetaType::Void, QMetaType::Int, QMetaType::Int,    3,    5,   
其中QMetaType::Void、QMetaType::Int都是定义在qmetatype.h的枚举。上面对应于：

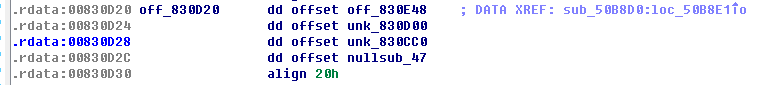
返回值类型, 第一个参数类型,第二个参数类型,第一个参数名称,第二个参数名称

参数名称和QMetaMethod.name一样，是在stringdata数组中的索引。  
如果类型没有在QMetaType中定义，如TSingapp\*这样的自定义类型，其类型将以

0x80000000 | 类型名索引

的形式定义，如例子中的TsignalApp\* mySlot2();被编译为：

0x80000000 | 9,

9即为  
QT\_MOC\_LITERAL(9, 66, 11) // "TsignalApp\*"  
QMetaMethod.flag保存相应方法的属性，如Public等，这里略过。  
至此，我们已经可以得到完整的QMetaMethod的函数签名信息了。  
确定QMetaObject.d位置  
以上对方法的解析前提是有QMetaObject.d的信息，下面来确定该位置。 实际上十分简单，Qt的元数据类型的签名为static const，也就是说只要在程序的.rdata段或者.data段查找如下形式的结构体即可：  
  
QMetaObject.d的主要特征是成员都是指什，superdata可以为null，因此找到连续三个的offset，且第4个成员为一函数指针，可以考虑为QMetaObject.d。  
  
破解Qt5程序的思路

在错误信息上下断点，其所在函数err\_func通常为槽。

断下后跟踪到对应的static\_metacall，确定err\_func的索引idx

解析QMetaObject.d，在d的data中查找idx对应的方法，解析方法的名称name

在字符串引用中查找name（可能要加上SLOT、SINGAL前缀）的引用，确定Qt进行connect的信号

查到对信号的引用，逆向完成破解

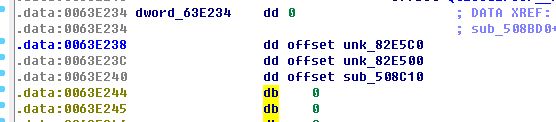
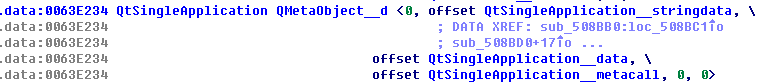
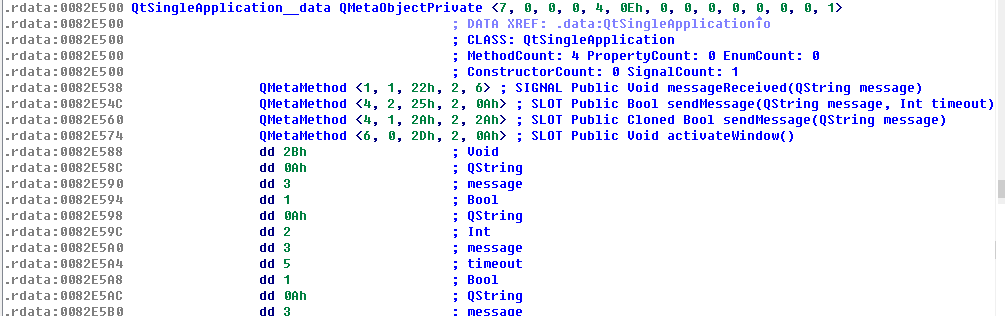
总结  
Qt通过信号/槽机制实现事件通信，可以在运行时动态connet。Qt在生成的可执行文件中保存了Qt对象（QObject）的元数据(QMetaObject.d结构），所有Qt对象均有一个static\_metacall函数，根据索引调用相应的方法。我们可以根据QMetaObject.d->data获得方法的名称、返回值、参数、索引等信息。  
本人写了一个IDA脚本，功能如下：

自动完成方法的名称、返回值、参数、索引的解析；

Qt结构体（QMetaObject.d、QArrayData、QMethod）的添加和标记；

static\_metacall函数的重命名；

支持32位和64位的Qt5程序

使用方法：  
将光标置于QMetaObject.d的起始处，Alt+F7调用本脚本即可。  
效果如下：  
  
运行脚本后：  
  
QMetaObject.d.data的效果如下：  
  
方法的名称和参数都还原了。