QT 原理2 信号槽的连接

存储结构

在`QObject`中，有个成员`d_ptr`

```
class Q_CORE_EXPORT QObject
{
protected:
    QScopedPointer<QObjectData> d_ptr;
};
```

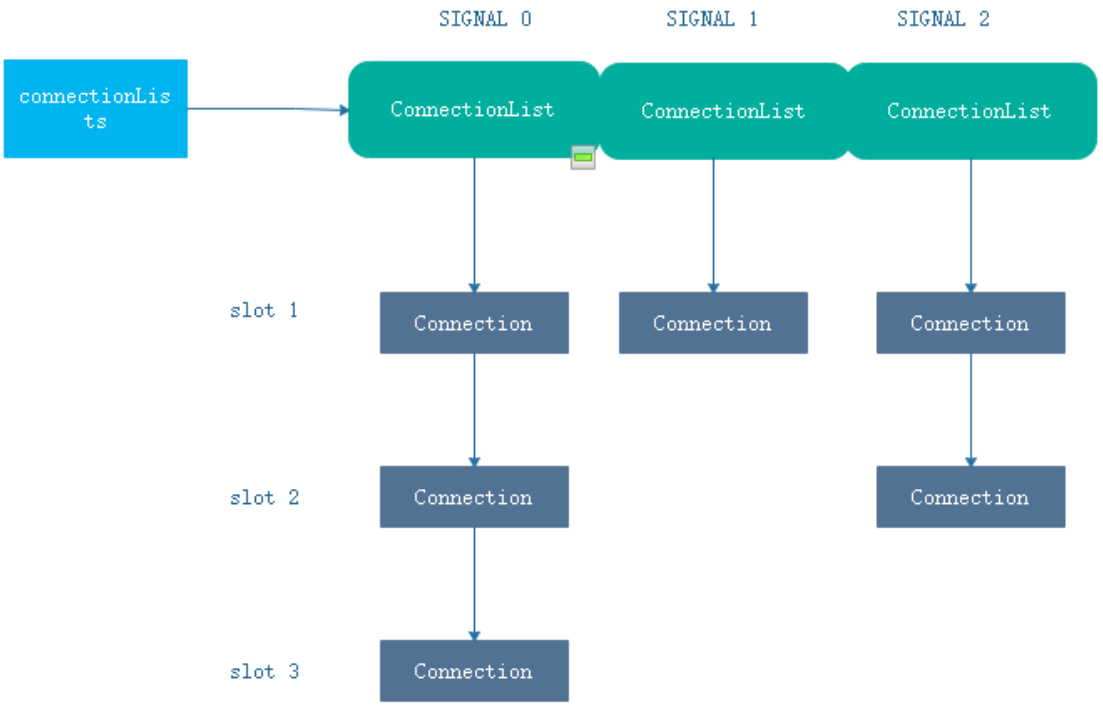
这个成员是个智能指针，指向的数据类型是`QObjectData`，实际上指向的是其子类`QObjectPrivate`，这个类被我删减了很多，目的为了使结构更加清晰，然后只突出主体结构：

```
class Q_CORE_EXPORT QObjectPrivate : public QObjectData
{
public:
    typedef void (*StaticMetaCallFunction)(QObject *, QMetaObject::Call, int, void **);
    //单向链表的节点，保存了信号的发送对象，接收对象以及接收对象中的接收的函数
    struct Connection
    {
        QObject *sender; //发送信号的对象
        QObject *receiver; //接受信号的对象
        StaticMetaCallFunction callFunction; //接受信号的对象的qt_static_metacall函数

        Connection *nextConnectionList; //下面的单向链表中指向下一个节点的指针
    };

    // 每个信号可以有多个槽函数连接，这个链表简单的认为是一个保存多个槽函数的单向链表
    struct ConnectionList {
        ConnectionList() : first(0), last(0) {}
        Connection *first; //链表的头节点
        Connection *last;  //链表的尾节点
    };

public:
    //保存信号的数组，每个信号的id为数组的下标索引
    QObjectConnectionListVector *connectionLists;
};
```

上面类中的注释写的很清楚，下面说一下整体的设计思路。

单个类

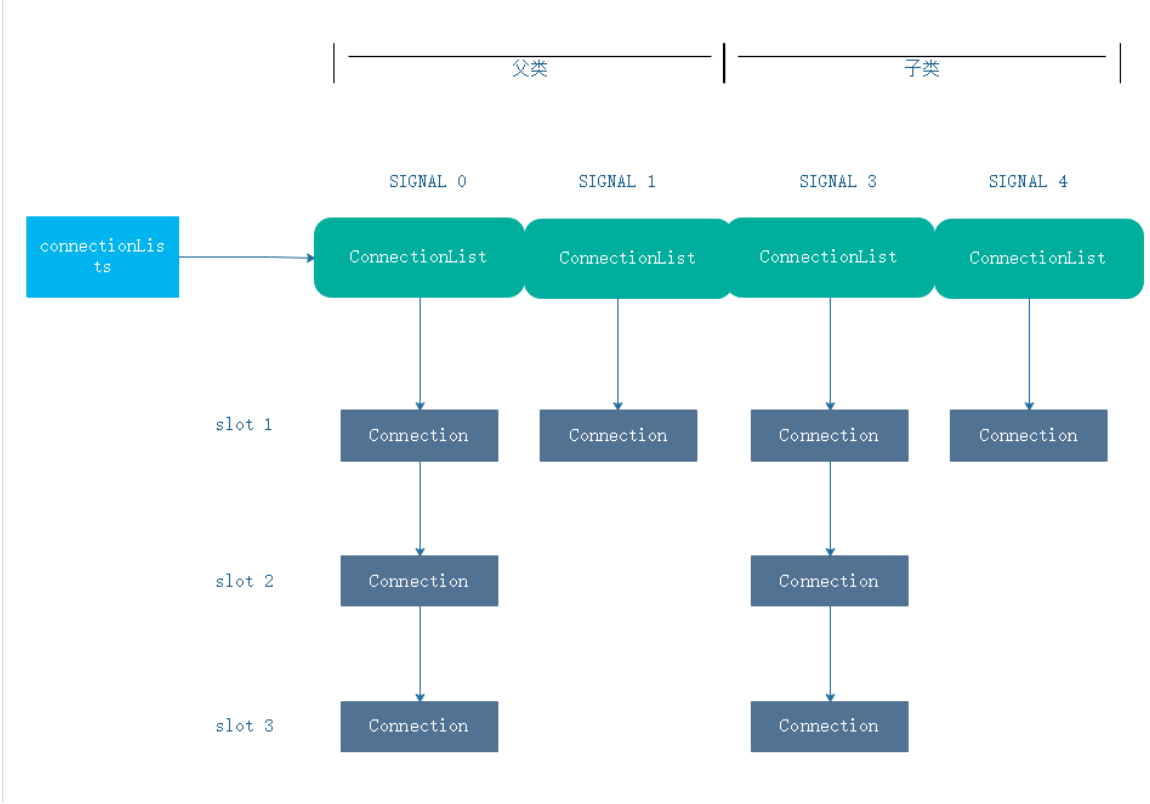
先拿单个类来说，因为一个信号可以有多个槽函数连接，所以对于单个信号来说，槽函数使用链表来存起来，这样当信号来的时候，遍历链表，并调用每个槽函数就可以了。但是对于一个类来说，可以有多个信号，QT使用数组存储信号，每个信号有个id，这个id就是数组中的下标索引。所以，实际上数组中元素是槽函数链表。



继承类

每个类都是有父类的，而父类也是有信号槽的，那么父类的信号槽该如何存储呢？
一种解决方案是每个类有自己的信号数组，但是这样做就需要把子类的数组和父类的数组联系起来，这样实现信号的查找，不过这样会比较麻烦。
第二种解决方案是，子类和父类共用一个信号数组，这样查找信号的时候就可以在同一个数组中查。不过这样会有另外一个问题，就是信号的id怎么分配？QT采用的办法是祖宗类的第一个信号id为0，

然后从祖宗类往子类依次增加。



信号的id

信号的id是值得说一个东西，它其实有个类内的id，还有一个全局id。全局id很简单，就是信号在信号数组中的id，那么类内id是什么呢？其实就是信号在静态成员`qt_meta_data_CStudent`中位于注释`signals`后面的索引值。

```
static const uint qt_meta_data_CStudent[] = {
 // signals: name, argc, parameters, tag, flags
       1,    1,   34,    2, 0x06 /* Public */, //SetAgeSig
       4,    2,   37,    2, 0x06 /* Public */, //SetSexSig

 // slots: name, argc, parameters, tag, flags
       7,    1,   42,    2, 0x0a /* Public */,
       8,    2,   45,    2, 0x0a /* Public */,
};
```

比如`SetAgeSig`是在`qt_meta_data_CStudent`的第一个信号，所以它的id为0，而`SetSexSig`的id为1，同样的槽函数也有类似的id。这个id值我称之为类内id，也叫做相对偏移id。为什么说这个呢，是因为当信号被发送的时候，使用的是类内id而非全局id

```
// SIGNAL 0
void CStudent::SetAgeSig(int _t1)
{
    void *_a[] = { nullptr, const_cast<void*>(reinterpret_cast<const void*>(&_t1)) };
    QMetaObject::activate(this, &staticMetaObject, 0, _a);
}
```

```
// SIGNAL 1
void CStudent::SetSexSig(bool _t1, int _t2)
{
    void *_a[] = { nullptr, const_cast<void*>(reinterpret_cast<const void*>(&_t1)),
const_cast<void*>(reinterpret_cast<const void*>(&_t2)) };
    QMetaObject::activate(this, &staticMetaObject, 1, _a);
}
```

而Qt则需要拿着这个类内id去找全局id，因为只有拿到全局id才可以从信号数组中拿到槽函数链表。

那么如何从类内id转换到全局id?
其实很简单，只要用这个类内id加上所有父类的信号的个数即可。

$$globalId = localId + countOfParentSigs$$

比如上图中，子类中有个信号X，它的类内id为0，父类信号的个数为2，则信号X的全局id为3（2+1=3）。

代码

自己跟踪`connect`函数和信号的发射函数即可。

其它资源

老外实现的信号槽

sigslot是信号槽的一个非常精炼的C++实现，作者是Sarah Thompson，sigslot实现只有一个头文件sigslot.h，跨平台且线程安全。在WebRTC中，sigslot .h是其基础的事件处理框架， 在多个模块的消息通知，响应处理中被使用。
sigslot库官网

boost库的信号槽