Les signaux et slots dans Qt 5.4

Les signaux-slots dans Qt 4

L'idée des signaux-slots est de créer un “lien” particulier entre deux fonctions de deux classes indépendantes, de façon à ce que lorsque l'on appelle la fonction du premier objet, la fonction du second objet est automatiquement appelée. La première fonction s'appelle “signal”, la seconde “slot”, le lien entre les deux s'appelle une “connexion”.

```
class A {
public:
    void fonction_de_A() const {}
};

class B {
public:
    void fonction_de_B() const {}
};

int main() {
    A un_objet_A;
    B un_objet_B;

    // création d'une connexion entre les deux fonction
    creer_connexion(un_objet_A, fonction_de_A, un_objet_B,
fonction_de_B);

    // lorsque l'on appelle la fonction de A, la fonction de
B est
    // automatiquement appelée
    un_objet_A.fonction_de_A(); // appelle
un_objet_B.fonction_de_B()
}
```

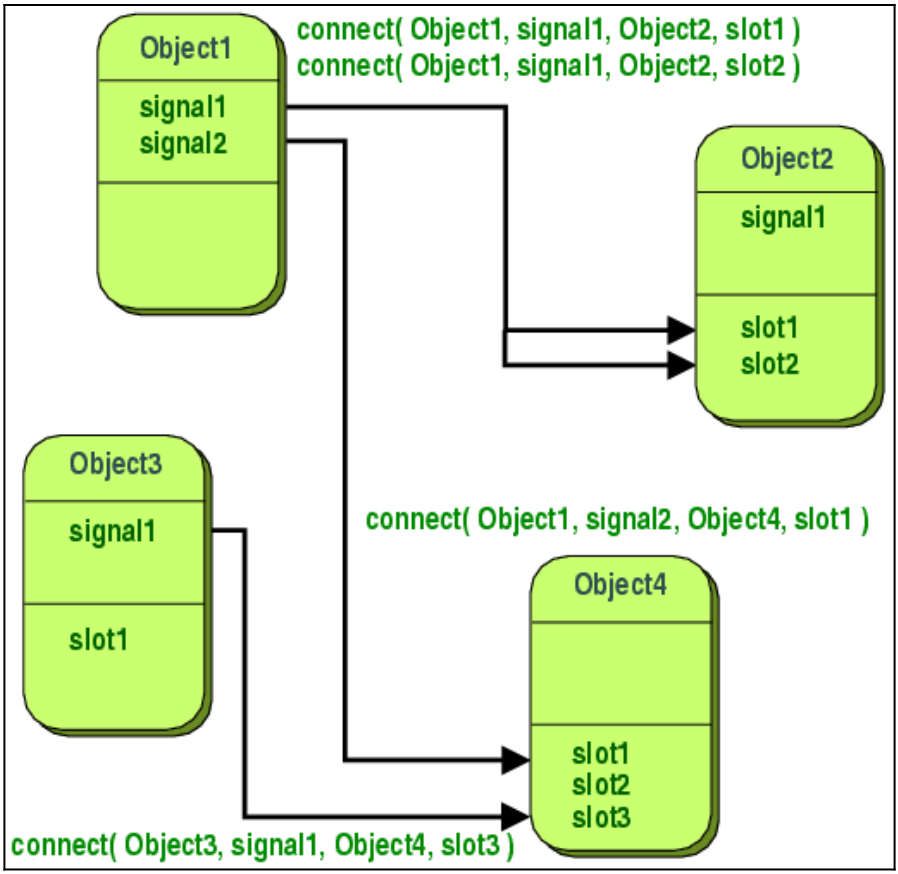
Avec ce type d'approche, vous pouvez utiliser autant de classes et de fonctions que vous souhaitez, celles-ci seront indépendantes. `A` n'a pas

besoin de connaitre `B` et `B` n'a pas besoin de connaitre `A`. On parle dans cette situation de “couplage faible”. Nous allons voir maintenant comme cela s'utilise dans Qt, mais sachez qu'il est également possible d'utiliser cette technique avec d'autres bibliothèques (par exemple, en utilisant Boost.Signals) ou en utilisant des fonctions comme argument (quelques mots-clés : std::function, functor object, lambda function, etc).

Le système de signaux et slots de Qt est relativement simple : lorsqu'un signal est émis avec le mot clé `emit`, tous les slots qui sont connectés à ce signal sont exécutés. Une connexion est créée en utilisant la fonction `QObject::connect`, en donnant les arguments suivants :

- un pointeur vers l'objet émetteur ;
- le nom du signal et la liste des types d'argument de la fonction signal (en utilisant la macro `SIGNAL`) ;
- un pointeur vers l'objet récepteur ;
- le nom du slot et la liste des types d'arguments de la fonction slot (en utilisant la macro `SLOT`).

Il est possible de connecter plusieurs signaux à un même slot, un signal à plusieurs slots ou un signal avec un signal.



La correspondance entre les arguments des signaux et slots est vérifiée comme une chaîne de caractères, lors de l'exécution. Ainsi, si vous connectez un `signal(float)` avec une `slot(double)`, la connexion sera refusée puisque les arguments ne correspondent pas, malgré le fait qu'un `float` peut être converti en `double` sans problème.

```
QAction* a = new QAction(this);
QWidget* w = new QWidget(this);
QObject::connect(
    a, SIGNAL(triggered()), // connecte le signal triggered()
de QAction
    w, SLOT(show())); // au slot show() de QWidget
// ce code permet donc d'afficher le QWidget lorsque
```

```
l'utilisateur active la QAction
```

Les classes de Qt fournissent de nombreux signaux et slots par défaut (la liste des signaux et slots des classes Qt est indiquée dans la documentation de Qt). Vous pouvez également créer vos propres signaux et slots dans vos classes, en respectant les règles suivantes :

- la classe doit dériver de `QObject` (directement ou indirectement) ;
- il faut appeler la macro `Q_OBJECT` au début de la classe ;
- les slots sont des fonctions classiques déclarées avec le mot-clé `slots` ;
- les signaux sont des fonctions non implémentées déclarées avec le mot-clé `signals`.

Concrètement, une classe doit ressembler à quelque chose comme cela :

```
#include <QObject>

class UneClasse : public QObject
{
    Q_OBJECT

public slots:
    void unSlot() { // code du slot // }

signals:
    void unSignal();
};
```

Comme le principe est de pouvoir connecter n'importe quelle classe et n'importe quels signaux et slots (à partir du moment où les paramètres de fonctions corresponds), vous pouvez sans problème connecter les classes Qt entre elles, avec vos propres classes ou vos propres classes entre elles.

Remarque importante : il faut obligatoirement déclarer les classes dérivant de `QObject` dans un fichier `.h` et non dans un fichier `.cpp`, sinon le pré-compilateur moc de Qt ne pourra pas fonctionner.

Les signaux et slots dans Qt 5

Dans Qt 4, il est possible de connecter uniquement les fonctions déclarées comme signaux et slots dans la classe, comme indiqué dans les codes d'exemple précédant. De plus, il faut que les déclarations des fonctions dans les macros `SIGNAL` et `SLOT` correspondent exactement, ce qui interdit l'utilisation de `typedef`/`using`, les espaces de noms ou les conversions implicites des types. Pour terminer, la connexion étant créé lors de l'exécution, il n'est pas possible de savoir dès la compilation s'il y a un problème, ce qui retard le diagnostic des problèmes.

Dans Qt 5, il est maintenant possible de connecter directement des pointeurs de fonctions ou d'utiliser des fonctions lambdas.

Connecter des pointeurs de fonctions membres

La connexion de pointeurs de fonctions est similaire à une connexion classique, en donnant un pointeur sur les objets et sur les fonctions. Les classes émettrices et réceptrices doivent dériver de QObject, mais il n'est pas nécessaire de déclarer les fonctions utilisées comme slots avec le mot clé `slots`.

```
class Sender : public QObject {
    Q_OBJECT

signals:
    void send(int i = 0);
};

class Receiver : public QObject {

public:
    void receive(int i = 0) { std::cout << "receive:" << i <
send(123);
};
```

```
QObject::connect(s, &Sender::send, r, &Receiver::receive);
```

L'avantage de cette écriture est que la compatibilité des paramètres est effectuée lors de la compilation et une conversion implicite est réalisée si nécessaire. Ainsi, il est possible de connecter un signal `mon_signal(float)` vers un slot `mon_slot(float)`, mais également vers un slot `mon_slot(double)`.

Lorsqu'il existe plusieurs fonctions surchargées, le compilateur ne sait pas à quelle fonction vous faites référence dans la connexion. Il faut donc donner explicitement la signature de la fonction, en effectuant une conversion. Par exemple, pour connecter les fonctions `QComboBox::currentIndexChanged(int)` ou `QComboBox::currentIndexChanged(QString)`, il est possible d'écrire :

```
QComboBox * comboBox = new QComboBox;

// utilisation de : void currentIndexChanged(int index)
connect(
    comboBox,
    static_cast<void (QComboBox::*)(int)>(&QComboBox::
currentIndexChanged),
    /* ... */);

// utilisation de : void currentIndexChanged(const QString &
text)
connect(
    comboBox,
    static_cast<void (QComboBox::*)(const QString &)>(&
QComboBox::currentIndexChanged),
    /* ... */);
```

(Cette astuce a été donnée par WinJérôme sur le forum de Developpez.com).

Connecter des lambdas, fonctions libres et

foncteurs

Il est également possible d'utiliser une fonction lambdas, fonctions libres et foncteurs comme slot dans une connexion. Par exemple, lorsque l'on souhaite connecter un signal `QPushButton::clicked` avec un slot `QLabel::setText(QString)` dans Qt 4, il n'est pas possible de créer directement la connexion.

```
QPushButton* button = new QPushButton;
QLabel* label = new QLabel;
QObject::connect(
    button, SIGNAL(clicked()),
    label, SIGNAL(setText(QString));
    // erreur, les paramètres de fonctions ne correspondent
pas
```

Il faut donc écrire un intermédiaire, par exemple :

```
class MyLabel : public QLabel { // ou une classe dérivée de
QPushButton
    Q_OBJECT
private:
    QPushButton* button;
    QLabel* label;
public:
    MyLabel(QObject* parent = 0) : QObject(parent) {
        button = new QPushButton(this);
        label = new QLabel(this);
        connect(button, SIGNAL(clicked()), SIGNAL(new_slot());
// Ok
    }
private slots:
    void new_slot() {
        emit label->setText("Mon texte");
    }
};
```

En utilisant les lambda, il est maintenant possible d'appeler directement :

```
QPushButton* button = new QPushButton(this);
QLabel* label = new QLabel(this);
```

```
QObject::connect(button, &QPushButton::clicked, [label](){
label->setText("Mon texte"); });
```

Dans ce code, on capture le pointeur vers l'objet récepteur et on appelle dans la fonction lambda la fonction `setText()`. Le résultat obtenu est identique au code précédant, mais il est possible de faire beaucoup d'autres choses dans la fonction lambda (par exemple déconnecter tous les signaux ou parcourir tous les enfants de l'objet récepteur).

Si le compilateur utilisé ne supporte pas les *variadic template*, les signaux et slots doivent avoir moins de 6 paramètres.

De la même façon, pour les fonctions libres et les foncteurs :

```
// un foncteur
class MyObject {
public:
    void operator()() { ... }
};

// une fonction libre
void foo() { ... }


MyObject* object = /* create object */;

QObject::connect(
    sender, unSignal(),
    object);

QObject::connect(
    sender, unSignal(),
    foo);
```

C++14

Le C++14 apporte quelques nouvelles syntaxes pour écrire des fonctions lambda : les fonctions lambda génériques et la capture étendue.

Ces syntaxes sont utilisables avec Qt 5.1 :

```
// lambda générique
connect(sender, &Sender::valueChanged, [=](const auto &
newValue) {
    receiver->updateValue("senderValue", newValue);
});

// capture étendue
connect(sender, &Sender::valueChanged, [receiver=getReceiver
()](const auto &newValue) {
    receiver->updateValue("senderValue", newValue);
});
```

Résumé des syntaxes utilisables

```
// Qt 4
QMetaObject::Connection QObject::connect(
    const QObject * sender, const char * signal,
    const QObject * receiver, const char * method,
    Qt::ConnectionType type = Qt::AutoConnection) [static]

QMetaObject::Connection QObject::connect(
    const QObject * sender, const char * signal,
    const char * method,
    Qt::ConnectionType type = Qt::AutoConnection) const

// Qt 4.8
QMetaObject::Connection QObject::connect(
    const QObject * sender, const QMetaMethod & signal,
    const QObject * receiver, const QMetaMethod & method,
    Qt::ConnectionType type = Qt::AutoConnection) [static]

// Qt 5.0
QMetaObject::Connection QObject::connect(
    const QObject * sender, PointerToMemberFunction signal,
    const QObject * receiver, PointerToMemberFunction method,
    Qt::ConnectionType type = Qt::AutoConnection) [static]

QMetaObject::Connection QObject::connect(
```

```
    const QObject * sender, PointerToMemberFunction signal,
    Functor functor) [static]

// Qt 5.2
QMetaObject::Connection QObject::connect(
    const QObject * sender, PointerToMemberFunction signal,
    const QObject * context, Functor functor,
    Qt::ConnectionType type = Qt::AutoConnection) [static]
```

Remarque : toutes ces fonctions sont `static`, sauf la seconde.

Conclusion

Vous pouvez télécharger un projet d'exemple montrant ces nouvelles fonctionnalités en action : la page de téléchargement sur GitHub.

Les images et codes d'exemple sont issus en partie de la documentation de Qt5 disponible à cette page : Signals & Slots.

Mises à jour

- 12/09/2014 : correction de la syntaxe des lambdas, suite à une remarque de WinJérôme.
- 27/10/2014 : mise à jour pour Qt 5.4 et suppression de la partie sur le couplage (cela fera partie d'un article plus généraliste sur la communication inter-classes).
- 27/11/2014 : ajout de la partie sur le C++14.

Sources

- Signals & Slots
- Differences between String-Based and Functor-Based

Connections
- Qt Developer Days 2013 - Olivier Goffart - Signals and Slots in Qt 5
- Les articles de woboq : Signals and Slots in Qt5, How Qt Signals and Slots Work et How Qt Signals and Slots Work - Part 2 - Qt5 New Syntax.
- C++14 : Le C++14 est arrivé et C++14 for Qt programmers.

Pour aller plus loin :

- La documentation des classes QSignalMapper, QSignalBlocker et QSignalSpy
- Benchmark for conception

Qt