Signaux et slots dans Qt 5

La nouvelle version de Qt vient de sortir en version 5 alpha. Cette version améliore la prise en charge de la nouvelle norme du C++, le C++11, et modifie ainsi le fonctionnement des signaux et slots de Qt. Cet article fait un rappel sur l'utilisation des signaux et slots et présente les nouvelles fonctionnalités offertes par Qt 5.

1 Couplage entre classes et intérêt des signaux et slots

Lorsque l'on souhaite faire communiquer des objets entre eux, il nécessaire en général que les objets se connaissent mutuellement. Par exemple :

```
class Receiver {
public:
    void slot() { cout << "slot exécuté" << slot(); }
};
class Sender {
public:
    void connect(Receiver* r) { receiver = r; }
    void signal() { receiver->slot(); }
private:
    Receiver* receiver;
};
int main() {
    // on crée nos objets
    Receiver receiver;
    Sender sender;
    // on connecte le sender et le receiver
    sender.connect(&receiver);
    // on émet le signal
    sender.signal();
}
```

Ce code présente cependant plusieurs problèmes :

- il faut que la classe `Sender` connaisse la classe `Receiver` et cela implique d'ajouter une dépendance (`include`) entre ces classes ;

- il est nécessaire de créer une fonction spécifique dans `Sender` pour chaque type de classe `Receiver` et pour chaque slot possible ;
- il ne permet pas de gérer des connexions vers plusieurs objets `Receiver` et il faut modifier le code pour gérer une liste de `Receiver`.

Ces contraintes s'accumulent dans un framework complexe comme Qt et cela alourdit fortement le code en ajoutant un nombre important de dépendances inutiles. Le code devient très vite ingérable [1].

Le système des signaux et slots permet de faire de s'affranchir de ces contraintes. Il est ainsi possible de faire communiquer des objets entre eux sans qu'il soit nécessaire que ces objets se connaissent mutuellement. On peut également choisir lors de la connexion le slot que l'on souhaite appeler lorsque le signal est émis. On passe ainsi d'un couplage fort (nécessité d'avoir une dépendance) à un couplage faible (plus de dépendance nécessaire) et l'on parle de découplage des classes.

2 Le système des signaux et slots dans Qt

Le système de signaux et slots de Qt est relativement simple. Lorsqu'un événement se produit, un signal est émis. Tous les slots qui sont connectés à ce signal sont alors exécutés. La fonction `QObject::connect` permet de créer une telle connexion. La forme la plus classique de cette fonction prend en paramètres un pointeur vers l'objet émetteur, le nom du signal (ainsi que la liste des types des arguments du signal), un pointeur vers l'objet récepteur et pour terminer le nom du slot (ainsi que la liste des types des arguments du slot). Il est possible de connecter plusieurs signaux à un même slot, un signal à plusieurs slots ou un signal avec un signal. Le compatibilité entre les classes et les signaux et slots est vérifiés lors de la compilation.

```
QAction a;
QWidget w;
QObject::connect(&a, SIGNAL(triggered()),
        // connecte le signal triggered() de QAction
        &w, SLOT(show()));
        // au slot show() de QWidget
```

1 Il est possible d'utiliser d'autres approches que celle présentée ici. En particulier, on peut utiliser les pointeurs de fonctions ou équivalents (callback). Le lecteur intéressé par la question pourra par exemple étudier l'approche utilisée dans Boost.Signals.

Les classes de Qt fournissent de nombreux signaux et slots par défaut. Ces signaux et slots seront disponibles dans les classes créées par les utilisateurs et dérivant des classes de Qt. Il est également possible de créer ses propres signaux et slots et de les connecter aux signaux et slots par défaut de Qt.

Remarque : il est nécessaire de mettre les déclaration des classes dérivées de `QObject` dans des fichiers d'en-tête et non dans un fichier d'implémentation, sinon, le système de pré-compilation de Qt ne pourra pas fonctionner.

```
#include  <QObject>

class Counter : public QObject {
    // on hérite de QObject pour bénéficier des méta-informations de Qt

    Q_OBJECT // cette macro permet de générer les
    // signaux et slots lors de la compilation

public slots:
    void setValue(int value) {
        if (value != m_value) { // lorsque la valeur est changée
            m_value = value;
            emit valueChanged(value); // on émet un signal valueChanged
        }
    }

signals:
    void valueChanged(int newValue);
    // signal émis lorsque la valeur est changée

private:
    int m_value;
};
```

Et dans le main.cpp :

```
int main() {
    Counter a, b;

    // on connecte valueChanged de a à setValue de b
    Qobject::connect(&a, &Counter::valueChanged,
                     &b, &Counter::setValue);
```

```
    a.setValue(12);
    // a émet un signal valueChanged qui active le slot setValue de b
    // a.value() == 12, b.value() == 12

    b.setValue(48);
    // b émet un signal valueChanged mais ce signal n'est pas connecté
    // à un slot : a.value() == 12, b.value() == 48
}
```

3 Créer une connexion dans Qt 5

Dans Qt 4, il est possible de connecter uniquement les fonctions déclarées comme signaux et slots dans la classes, comme indiqué dans le code d'exemple précédant. Dans Qt 5, il est maintenant possible de connecter directement des pointeurs de fonctions ou d'utiliser des fonctions lambdas.

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 slots avec le mot clé `slots`.

```
class Sender : public QObject {
    Q_OBJECT
signals:
    void send(int i = 0);
};
class Receiver : public QObject {
    Q_OBJECT
public:
    void receive(int i = 0) {
        qDebug() << "Receiver.receive :" << i;
    }
};
// Utilisation avec les pointeurs de fonctions
QObject::connect(&sender, &Sender::send,
                 &receiver, &Receiver::receive);
emit sender.send(2);
sender.disconnect();
```

L'avantage de cette écriture est que la compatibilité des paramètres est effectuée lors de la compilation et non lors de l'exécution.

Pour les fonctions lambdas :

```
// Utilisation avec les fonctions lambdas
QObject::connect(&sender, &Sender::send,
                 [&receiver](int i = 0){ receiver.receive(i); });
emit sender.send(3);
```

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

Si le compilateur utilisé ne support pas les variadics templates, les signaux et slots doivent avoir moins de 6 paramètres.

4 Remarques

Vous pouvez télécharger un projet d'exemple montrant ces nouvelles fonctionnalités en action : la page de téléchargement. Les images et codes d'exemple sont issus de la documentation de Qt5 disponible à cette page : Qt 5.0: Signals & Slots.