2. Gestion du temps

Certaines applications demandent d'exécuter des actions au bout d'un certain délai ou de les répéter à des intervalles réguliers. La classe `QTimer` de Qt est une classe qui permet de créer des minuteries (timers). Elle est couramment utilisée pour gérer les événements temporisés dans les applications Qt, comme les mises à jour périodiques d'interface utilisateur, les animations ou encore, la gestion de tâches en arrière-plan.

2.1. Création d'un QTimer

Un objet `QTimer` peut être créé avec la méthode classique :

```
QTimer *timer = new QTimer(this);
```

`this` est le parent de l'objet timer, ce qui permet de gérer sa destruction automatique avec le parent. Bien souvent, il est attribut d'une classe et instancié dynamiquement dans le constructeur. De même, rien n'empêche de créer une instance simple dans la classe.

2.2. Définition d'un intervalle

L'intervalle, exprimé en millisecondes, détermine la fréquence à laquelle le timer déclenche son signal. Pour définir l'intervalle :

```
timer->setInterval(1000); // Déclenche toutes les 1000 ms (1 seconde)
```

Cette méthode n'est pas obligatoire, la période de déclenchement peut être définie lors du lancement du timer.

2.3. Lancement et arrêt du timer

Deux approches sont donc possibles pour lancer un timer, soit en utilisant l'intervalle prédéfini précédemment, soit en précisant une nouvelle période.

```
timer->start(); // Utilise l'intervalle défini précédemment
ou
timer->start(500); // Déclenche toutes les 500 ms
```

Par défaut, le timer est relancé à la fin de la période. pour arrêter un timer en cours

```
timer->stop();
```

Si le timer ne doit se déclencher qu'une seule fois, une autre manière de procéder consiste à utiliser la méthode QTimer::singleShot :

```
timer->setSingleShot(true);
timer->start(2000); // S'exécute une seule fois après 2 secondes
```

2.4. Signal `timeout()`

Le signal timeout() est émis chaque fois que l'intervalle défini expire. Il est nécessaire de le connecter à un slot réalisant le traitement approprié.

```
connect(timer, &QTimer::timeout, this, &MaClasse::onTimerTimeout);
```

La classe MaClasse doit posséder le slot onTimerTimeout et doit hériter de la classe QObject.

Exemple d'utilisation

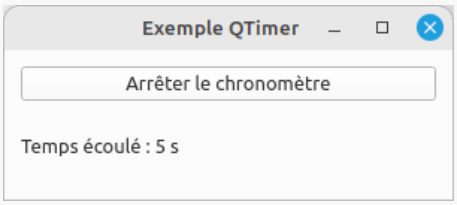
Exemple d'utilisation de QTimer

```
#include <QApplication>
#include <QWidget>
#include <QPushButton>
#include <QLabel>
#include <QVBoxLayout>
#include <QTimer>
class TimerExample : public QWidget {
    Q_OBJECT
public:
    TimerExample(QWidget *parent = nullptr) : QWidget(parent) {
        // Création des widgets
        button = new QPushButton("Démarrer le chronomètre", this);
        timeLabel = new QLabel("Temps écoulé : 0 s", this);
        // Création du QTimer
        timer = new QTimer(this);
        // Mise en place du layout
        QVBoxLayout *layout = new QVBoxLayout(this);
        layout->addWidget(button);
        layout->addWidget(timeLabel);
        setLayout(layout);
        // Connexion du bouton pour démarrer ou arrêter le timer
        connect(button, &QPushButton::clicked, this,
                        &TimerExample::startStopTimer);
        // Connexion du timer au slot pour mettre à jour le label
        connect(timer, &QTimer::timeout, this, &TimerExample::onTimer_timeout);
    }
private slots:
    void startStopTimer() {
        if (timer->isActive()) {
            timer->stop();
            button->setText("Démarrer le chronomètre");
        } else {
            elapsedSeconds = 0; // Réinitialiser le temps écoulé
            timeLabel->setText("Temps écoulé : 0 s");
            timer->start(1000);
            button->setText("Arrêter le chronomètre");
        }
    }
    void onTimer_timeout() {
        // Incrémenter le nombre de secondes écoulées
        elapsedSeconds++;
        // Mettre à jour le label avec le nouveau temps écoulé
        timeLabel->setText(QString("Temps écoulé : %1 s").arg(elapsedSeconds));
    }
private:
    QPushButton *button;
    QLabel *timeLabel;
    QTimer *timer;
    int elapsedSeconds = 0; // Compteur pour le temps écoulé en secondes
};
```

```
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    TimerExample window;
    window.setWindowTitle("Exemple QTimer");
    window.resize(300, 100);
    window.show();

    return app.exec();
}
```



2.5. Fonctionnalités supplémentaires

- `QTimer::remainingTime()` : Retourne le temps restant avant le prochain déclenchement.
- `QTimer::isActive()` : Indique si le timer est en cours d'exécution.
- `QTimer::interval()` : Retourne l'intervalle actuellement défini.

2.6. Gestion de la précision

La précision du timer reste dépendante du comportement de la boucle de message et du système d'exploitation. Cela signifie que le comportement peut encore varier selon la plateforme (Windows, macOS, Linux).

La bibliothèque Qt a introduit des améliorations pour permettre un contrôle plus fin de la précision. Ainsi depuis Qt 5, il est possible de définir explicitement la précision du timer grâce à la méthode `QTimer::setTimerType()`. Il existe trois types de timers :

- **Qt::CoarseTimer** (précision approximative) : Convient pour les actions non critiques en termes de temps, car il utilise une faible consommation de CPU.
- **Qt::PreciseTimer** (précision élevée) : Utilisé pour les actions nécessitant une grande précision.
- **Qt::VeryCoarseTimer** (précision encore plus approximative) : Utilisé pour des tâches de fond où la précision est peu importante.

Pour définir un timer précis :

Définition d'un timer précis

```
QTimer *timer = new QTimer(this);
timer->setTimerType(Qt::PreciseTimer);
timer->start(1000); // Démarre un timer avec une haute précision à 1 seconde
```

La précision du timer reste limitée par l'OS et le matériel. Par exemple, sur certaines plateformes, un intervalle de 1 ms peut ne pas être respecté à la lettre en raison des limitations du système.

2.7. Alternatives pour une meilleure précision

Si une précision extrême est nécessaire, au-delà de ce que peut fournir `QTimer`, d'autres options sont envisageables :

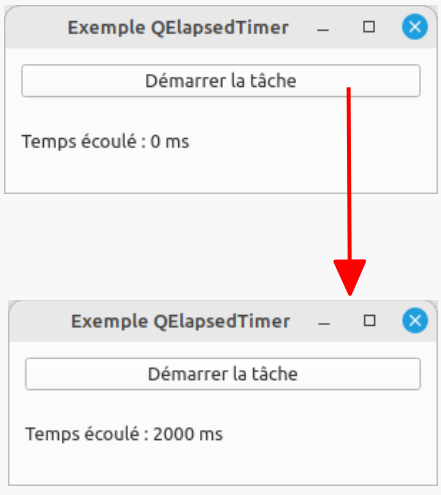
- **`QElapsedTimer`** Pour mesurer précisément des intervalles de temps écoulés (sans dépendre d'un timer périodique), vous pouvez utiliser `QElapsedTimer`. C'est un bon moyen pour mesurer des intervalles courts avec une grande précision :

Alternative

```
QElapsedTimer timer;
timer.start();
// ... exécuter du code ...
qint64 elapsed = timer.elapsed(); // Temps écoulé en millisecondes
```

- **Chronomètres de bas niveau (si applicable)** Si vous avez besoin d'une précision encore meilleure (microsecondes), il est parfois nécessaire d'utiliser des API spécifiques au système d'exploitation (comme `QueryPerformanceCounter` sur Windows ou `clock_gettime` sur Linux).

Exemple complet :

Exemple d'utilisation de QElapsedTimer

```
#include <QApplication>
#include <QWidget>
#include <QPushButton>
#include <QLabel>
#include <QVBoxLayout>
#include <QElapsedTimer>
#include <QThread>  // Pour simuler une tâche longue avec QThread::sleep
class TimerExample : public QWidget {
   Q_OBJECT
public:
    TimerExample(QWidget *parent = nullptr) : QWidget(parent) {
        // Création des widgets
        button = new QPushButton("Démarrer la tâche", this);
        timeLabel = new QLabel("Temps écoulé : 0 ms", this);
        // Mise en place du layout
        QVBoxLayout *layout = new QVBoxLayout(this);
        layout->addWidget(button);
        layout->addWidget(timeLabel);
        setLayout(layout);
        // Connexion du bouton à la méthode qui lance la tâche
        connect(button, &QPushButton::clicked, this, &TimerExample::startTask);
    }
private slots:
    void startTask() {
        // Création du chronomètre
        QElapsedTimer timer;
        timer.start();  // Démarrage du chronomètre
        // Simuler une tâche longue (ici, une pause de 2 secondes)
        QThread::sleep(2);  // Pause de 2 secondes pour simuler une tâche longue
        // Calcul du temps écoulé
        qint64 elapsed = timer.elapsed();  // Temps en millisecondes
        // Affichage du temps dans le QLabel
        timeLabel->setText(QString("Temps écoulé : %1 ms").arg(elapsed));
    }
private:
    QPushButton *button;
    QLabel *timeLabel;
};

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    TimerExample window;
    window.setWindowTitle("Exemple QElapsedTimer");
    window.resize(300, 100);
    window.show();
    return app.exec();
}
```