QCoro
Корутины в Qt

Илья Быконя
АО ВНИИЖТ

Проблемы отсутствия корутин

1. Callback hell
2. Копирование контекста
3. Элегантность решений

```
auto manager = new QNetworkAccessManager{};
auto reply_1 = manager->get(QNetworkRequest{ QUrl{ "..." } });
QObject::connect(reply_1, &QNetworkReply::finished,
[reply_1, manager] {
    const auto result_1 = reply_1->readAll();
    /* ... */
    auto reply_2 = manager->get(QNetworkRequest{ QUrl{ "..." } });
    QObject::connect(reply_2, &QNetworkReply::finished,
    [reply_1, reply_2, manager] {
        const auto result_1 = reply_2->readAll();
        /* ... */
        reply_1->deleteLater();
        reply_2->deleteLater();
        manager->deleteLater();
    });
});
```

```
QNetworkAccessManager manager{};
auto reply_1 = QScopedPointer{ co_await manager.get(QNetworkRequest{ QUrl{ "..." } }) };
const auto result_1 = reply_1->readAll();
/* ... */
auto reply_2 = QScopedPointer{ co_await manager.get(QNetworkRequest{ QUrl{ "..." } }) };
const auto result_2 = reply_2->readAll();
/* ... */
```

QCoro future

Операции	QCoro::Task<T>	QCoro::Generator<T>	QCoro::AsyncGenerator<T>
co_return	+	-	-
co_yield	-	+	+
co_await	+	-	+

QCoro::Task<T>

```
QCoro::Task<> processFor() {
    QTimer timer{};
    timer.start(1000);
    for(auto index = 0; index < 10; ++index, co_await timer) {
        qDebug() << "He-he: " << index;
    }
}
```

QCoro::Task<T>

```
QCoro::Task<> processFor() {
    for(auto index = 0; index < 10; ++index, co_await QCoro::sleepFor(1s)) {
        qDebug() << "He-he: " << index;
    }
}
```

QCoro::Task<T>

```
QCoro::Task<int32_t> generate_number() {
    constexpr auto request =
"https://www.random.org/integers/?num=1&min=1&max=100&col=1&base=10&format=plain&rnd=new";

    QNetworkAccessManager manager{};
    auto reply = QScopedPointer{ co_await manager.get(QNetworkRequest{ QUrl{ request } }) };
    co_return reply->readAll().toInt();
}

/*...*/

generate_number().then([](int32_t number) { qDebug() << number; });
```

QCoro::Generator<T>

```
QCoro::Generator<double> values() {
    for(auto index = 0; index < 100; ++index) {
        qDebug() << "Return coro value: " << index;
        co_yield index;
    }
}

/*...*/

for(auto value: values())
    qDebug() << "Use coro value: " << value;
```

QCoro::AsyncGenerator<T>

```
//Обычный цикл for для iterable-объекта
auto iterable = /*...*/;
for(auto iter = iterable.begin(); iter != iterable.end(); ++iter) {}

//Цикл for для QAsyncGenerator итератора
QCoro::AsyncGenerator<double> iterable = /*...*/;
for(auto iter = co_await iterable.begin(); iter != iterable.end(); co_await ++iter) {}
```

QCoro::AsyncGenerator<T>

```
constexpr auto request =
"https://www.random.org/integers/?num=1&min=1&max=100&col=1&base=10&format=plain&rnd=new";

QCoro::AsyncGenerator<int32_t> generate_numbers() {
    QNetworkAccessManager manager{};
    for(;;) {
        auto reply = QScopedPointer{ manager.get(QNetworkRequest{ QUrl{ request } }) };
        co_await qCoro(reply.get(), &QNetworkReply::finished);
        co_yield reply->readAll().toInt();
    }
}
/*...*/
auto generator = generate_numbers();
for(auto iter = co_await generator.begin(); iter != generator.end(); co_await ++iter) {
    qDebug() << "Coro value: " << *iter;
}
```

Утечка памяти

```
QCoro::Task<int32_t> task() {
    co_await QCoro::sleepFor(std::chrono::years{ 4 });
    co_return 42;
}
```

При удалении QCoro::Generator и QCoro::AsyncGenerator память, выделенная под корутину, высвобождается.

Память, выделенная для QCoro::Task, высвобождается только после операции co_return;

Ожидание по сигналу

```
class CustomObject: public QObject {
    Q_OBJECT
public:
    explicit CustomObject(QObject *parent = nullptr);
signals:
    void completed_zero();
    void completed_one(int32_t);
    void completed_two(int32_t, bool);
    void completed_three(int32_t, bool, char);
protected:
    void timerEvent(QTimerEvent *event);
};
```

Ожидание по сигналу

```
CustomObject::CustomObject(QObject *parent)
    :QObject{ parent } { this->startTimer(std::chrono::seconds{ 5 }); }

void CustomObject::timerEvent(QTimerEvent *event) {
    QObject::timerEvent(event);
    emit this->completed_zero();
    emit this->completed_one(42);
    emit this->completed_two(42, true);
    emit this->completed_three(42, true, '@');
}
```

Ожидание по сигналу

```
QCoro::Task<> awaitCustomObject() {
    CustomObject object{};
    const auto r0 = co_await qCoro(&object, &CustomObject::completed_zero);
    const auto r1 = co_await qCoro(&object, &CustomObject::completed_one);
    const auto r2 = co_await qCoro(&object, &CustomObject::completed_two);
    const auto r3 = co_await qCoro(&object, &CustomObject::completed_three);
}

// r0 — tuple<>
// r1 — int32_t
// r2 — tuple<int32_t, bool>
// r3 — tuple<int32_t, bool, char>
```

Ожидание по сигналу

```
QCoro::Task<> awaitCustomObject() {
    CustomObject object{};
    const auto r0 = co_await qCoro(&object, &CustomObject::completed_zero);
    const auto r1 = co_await qCoro(&object, &CustomObject::completed_one);
    const auto [r2_1, r2_2] = co_await qCoro(&object, &CustomObject::completed_two);
    const auto [r3_1, r3_2, r3_3] = co_await qCoro(&object, &CustomObject::completed_three);
}
```

QML Task

```
class RandomNumberUploader: public QObject {
private:
    Q_OBJECT
public:
    RandomNumberUploader(QObject* parent = nullptr);
    Q_INVOKABLE QCoro::QmlTask generate() const;
private:
    QCoro::Task<int32_t> generate_number() const;
};
```

QML Task

```
RandomNumberUploader::RandomNumberUploader(QObject* parent)
    :QObject{ parent } {}

QCoro::QmlTask RandomNumberUploader::generate() const {
    return generate_number();
}

QCoro::Task<int32_t> RandomNumberUploader::generate_number() const {
    constexpr auto request =
        "https://www.random.org/integers/?num=1&min=1&max=100&col=1&base=10&format=plain&rnd=new";
    qDebug() << "Request for upload value";
    QNetworkAccessManager manager{};
    auto reply = QScopedPointer{ co_await manager.get(QNetworkRequest{ QUrl{ request } }) };
    const auto value = reply->readAll().toInt();
    qDebug() << "Uploaded value: " << value;
    co_return value;
}
```

QML Task

```
import QCoro 0
import app.modules 1.0

//...

RandomNumberUploader{ id: uploader }
Row {
    Button {
        text: "dada"
        onClicked: label.provider = uploader.generate().await()
    }
    Text {
        id: label
        property var provider: uploader.generate().await()
        text: provider.value
    }
}
```

QML Image Provider

Image { source: "image://provider_id/some/image/id/he-he" }

image:// — Qt-шный протокол, чтобы лазать не куда-то, а к image provider-ам
provider_id — id провайдера, указанный при регистрации
some/image/id/he-he — id изображения

QML Image Provider

```
Row {
    Column {
        Image { source: "image://standard_images/yellow" }
        Image { source: "image://standard_images/red" }
    }
    Column {
        Image { source: "image://async_images/blue" }
        Image { source: "image://async_images/cyan" }
    }
    Column {
        Image { source: "image://coro_images/green" }
        Image { source: "image://coro_images/magenta" }
    }
}
```

QML Image Provider. Standard

```
class StandardImageProvider: public QQuickImageProvider {
public:
    StandardImageProvider();
    virtual QPixmap requestPixmap(const QString &id, QSize *size, const QSize &requestedSize) override;
};
//===============================================

StandardImageProvider::StandardImageProvider()
        :QQuickImageProvider(QQuickImageProvider::Pixmap){}

QPixmap StandardImageProvider::requestPixmap(const QString &id, QSize *size, const QSize &requestedSize) {
    if (size) { *size = QSize{ 100, 50 }; }
    QPixmap pixmap{ 100, 50 };
    pixmap.fill(QColor(id).rgba());
    return pixmap;
}
```

QML Image Provider. Async

```
class AsyncImageProvider : public QQuickAsyncImageProvider {
public:
    virtual QQuickImageResponse *requestImageResponse(const QString &id, const QSize &requestedSize) override;
};

//================================================

QQuickImageResponse *AsyncImageProvider::requestImageResponse(const QString &id, const QSize &requestedSize) {
    return new AsyncImage{ id, requestedSize };
}
```

QML Image Provider. Async

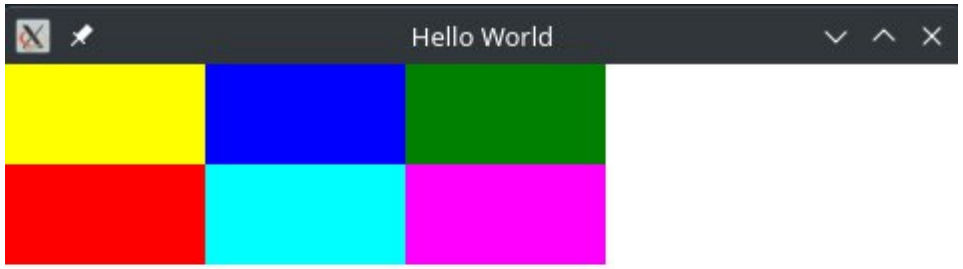
```
class AsyncImage: public QQuickImageResponse {
private:
    QImage m_image{};
public:
    AsyncImage(const QString &id, const QSize &requestedSize);
    virtual QQuickTextureFactory *textureFactory() const override;
};
//============================================

AsyncImage::AsyncImage(const QString &id, const QSize &requestedSize) {
    QTimer::singleShot(2500, this, &AsyncImage::finished);
    m_image = QImage{ QSize{ 100, 50 }, QImage::Format_RGB32 };
    m_image.fill(QColor(id).rgba());
}
QQuickTextureFactory *AsyncImage::textureFactory() const {
    return QQuickTextureFactory::textureFactoryForImage(m_image);
}
```

QML Image Provider. Coro

```
class CoroImageProvider: public QCoro::ImageProvider {
public:
    virtual QCoro::Task<QImage> asyncRequestImage(const QString &id, const QSize &requestedSize) override;
};

//================================================

QCoro::Task<QImage> CoroImageProvider::asyncRequestImage(const QString &id, const QSize &requestedSize) {
    const auto color = QColor(id).rgba();

    co_await QCoro::sleepFor(5s);
    auto image = QImage{ 100, 50, QImage::Format_RGB32 };
    image.fill(color);
    co_return image;
}
```

QML Image Provider. Регистрация

```
QQmlApplicationEngine engine{};
engine.addImageProvider("standard_images", new StandardImageProvider{});
engine.addImageProvider("async_images", new AsyncImageProvider{});
engine.addImageProvider("coro_images", new CoroImageProvider{});
```



QML Image Provider. Http

```
class HttpCoroImageProvider: public QCoro::ImageProvider {
public:
    virtual QCoro::Task<QImage> asyncRequestImage(const QString &id, const QSize &requestedSize) override;
};

//================================================

QCoro::Task<QImage> HttpCoroImageProvider::asyncRequestImage(const QString &id, const QSize &requestedSize) {
    auto manager = QNetworkAccessManager{};
    const auto reply = QScopedPointer{
        co_await manager.get(QNetworkRequest{ QUrl{
            QString{ "http://<domain.com>/images/%1.png" }.arg(id) } })
    };
    co_return QImage::fromData(reply->readAll());
}
```

QML Image Provider. Http