Developer Days 2013

Asynchronous programming (and more) with Qt5 and C++11

Dario Freddi, Ispirata

Qt Developer Days 2013

Hello

ispirata

Hello

Hello

Hello

Qt5 <3 C++11

C++11 in 10 minutes

A quick tour

C++11 in 10 minutes

Main Concepts

C++11 in 10 minutes

```
class A {
protected:
    virtual int myMethod(int arg, char *args);
};

class B : public A {
protected:
    virtual void myMethod(int arg, char *args);
};
```

C++11 in 10 minutes

```
class A {
protected:
    virtual int myMethod(int arg, char *args);
};

class B : public A {
protected:
    void myMethod(int arg, char *args) override;
};
```

C++11 in 10 minutes

```
class A {
protected:
    virtual int myMethod(int arg, char *args) final;
};

class B : public A {
protected:
    virtual int myMethod(int arg, char *args);
};
```

C++11 in 10 minutes

```
class A {
protected:
    void myMethod(int, char *) Q_DECL_FINAL;
};

class B : public A {
protected:
    void myMethod(int, char *) Q_DECL_OVERRIDE;
};
```

C++11 in 10 minutes

```
#define MULTIPLY(a, b) a*b
```

C++11 in 10 minutes

```
#define MULTIPLY(a, b) a*b

constexpr int multiply(int a, int b) {
    return a*b;
}
```

C++11 in 10 minutes

Developer Days 2013

```
constexpr int factorial (int n)
{
    return n > 0 ? n * factorial( n - 1 ) : 1;
}
```

C++11 in 10 minutes

```
class Stuff
{
    public:
    constexpr Stuff (int x, int y) : m_x( x ),m _y( y ) {}
    constexpr double compute()
    {
        return m_x * m_y * 42;
    }
    private:
        int m_x;
        int m_y;
};
```

C++11 in 10 minutes

```
Q_DECL_CONSTEXPR int factorial (int n)
{
    return n > 0 ? n * factorial( n - 1 ) : 1;
}
```

C++11 in 10 minutes

```
Q_FOREACH(const QString &element, list) {
    // code, code, code...
}
```

C++11 in 10 minutes

```
for (const QString &element : list) {
    // code, code, code...
}
```

C++11 in 10 minutes

```
enum Stuff {
    BEANS,
    QT,
    OTHER
};

enum MoarStuff {
    PILLOWS,
    ONIONS,
    OTHER
};
```

C++11 in 10 minutes

```
enum class Stuff {
    BEANS,
    QT,
    OTHER
};

enum class MoarStuff {
    PILLOWS,
    ONIONS,
    OTHER
};
```

C++11 in 10 minutes

```
enum class Stuff;

void wheresMyStuff(Stuff stuff);

enum class Stuff : int {
    BEANS,
    QT,
    OTHER
};
```

C++11 in 10 minutes

```
void doStuff(int);
void doStuff(char *);

doStuff(0);
```

C++11 in 10 minutes

```
void doStuff(int);
void doStuff(char *);

doStuff(nullptr);
```

C++11 in 10 minutes

```
void doStuff(int);
void doStuff(char *);

doStuff(Q_NULLPTR);
```

C++11 in 10 minutes

Developer Days 2013

The main course

Lambdas

```
auto areLambdasAmazing =
    [this] -> bool { return true; }
```

Lambdas

```
return QQmlListProperty<MyObject>(this, 0,
        [] (QQmlListProperty<MyObject> *list, Object *m) {
            Controller *c = qobject_cast<Controller *>(list->object);
            if (m) {
                c->append(m);
            }
         },
         [] (QQmlListProperty<MyObject> *list) -> int {
            return qobject_cast<Controller *> (list->object)->count();
         },
         [] (QQmlListProperty<MyObject> *list, int at) -> Object* {
            return qobject_cast<Controller *>(list->object)->at(at);
         },
         [] (QQmlListProperty<MyObject> *list) {
            Controller *c = qobject_cast<Controller *>(list->object);
            c->clearObjects();
         });
```

Lambdas

```
connect(myobj, &QObject::destroyed,
        [this] { qDebug() << “oh noes!”; });
```

Lambdas in Qt 5.1

- Tied to the context of the sender
- Hence, tied to Qt::DirectConnection
- Little control over the connect mechanism

Lambdas

```
connect(jobManager, &JobManager::jobRemoved,
              [this] (uint id) {
             if (d->id == id) {
                 sendSignalAndDestroyObject();
             }
});
```

Lambdas

What is happening:

- The connection is stored in a QMetaObject::Connection
- The sender (context) is never destroyed
- The connection will be still alive when this is destroyed

Lambdas

```
connect(jobManager, &JobManager::jobRemoved,
              [this] (uint id) {
             if (d->id == id) {
                 sendSignalAndDestroyObject();
             }
});
```

Lambdas in Qt 5.1

- Can be tied to a different QObject context
- Functor connect behaves as standard connect does (Qt::AutoConnection)

Lambdas

```
connect(jobManager, &JobManager::jobRemoved,
            this, [this] (uint id) {
            if (d->id == id) {
                sendSignalAndDestroyObject();
            }
});
```

Lambdas

```
connect(object, &Object::randomSignal,
            objectInDifferentThread, [this, object] {
            if (QThread::currentThread() !=
                object->thread()) {
                // This is definitely not going to happen!
            }
});
```

Under the hood

Variadic templates in QObject::connect's implementation

Under the hood

```
template <typename Func1, typename Func2>
    static inline typename
QtPrivate::QEnableIf<int(QtPrivate::FunctionPointer<Func2>::ArgumentCount)
>= 0 && !QtPrivate::FunctionPointer<Func2>::IsPointerToMemberFunction,
QMetaObject::Connection>::Type

            connect(const typename QtPrivate::FunctionPointer<Func1>::Object
*sender, Func1 signal, const QObject *context, Func2 slot,
                      Qt::ConnectionType type = Qt::AutoConnection)
```

Under the hood

```
template<class Obj, typename Ret, typename Arg1> struct
FunctionPointer<Ret (Obj::*) (Arg1)>
    {
        typedef Obj Object;
        typedef List<Arg1, void> Arguments;
        typedef Ret ReturnType;
        typedef Ret (Obj::*Function) (Arg1);
        enum {ArgumentCount = 1, IsPointerToMemberFunction = true};
        template <typename Args, typename R>
        static void call(Function f, Obj *o, void **arg) {
            (o->*f)((*reinterpret_cast<typename RemoveRef<typename
Args::Car>::Type *>(arg[1]))), ApplyReturnValue<R>(arg[0]);
        }
    };
```

Under the hood

```
template<class Obj, typename Ret, typename... Args> struct
FunctionPointer<Ret (Obj::*) (Args...)>
    {
        typedef Obj Object;
        typedef List<Args...>  Arguments;
        typedef Ret ReturnType;
        typedef Ret (Obj::*Function) (Args...);
        enum {ArgumentCount = sizeof...(Args),
IsPointerToMemberFunction = true};
        template <typename SignalArgs, typename R>
        static void call(Function f, Obj *o, void **arg) {
            FunctorCall<typename Indexes<ArgumentCount>::Value,
SignalArgs, R, Function>::call(f, o, arg);
        }
    };
```

Lambdas

Initializing a chain of asynchronous objects

Async initialization chains

```
class AsyncInitObject : public QObject {
    Q_OBJECT
public:
    void init();
protected:
    virtual void initImpl() = 0;
    void setReady(bool status);
signals:
    void ready();
    void error();
};
```

Async initialization chains

```
{
    connect(anObject, SIGNAL(ready()),
                SLOT(initNext()));
}

{
    prepare();
    connect(otherObject, SIGNAL(ready()),
                SLOT(initNextAgain()));
}
```

Async initialization chains

```
{
    connect(anObject, &AsyncInitObject::ready,
                [this, otherObject] {
                    prepare();
                    otherObject->init();
                    connect(otherObject, &ASIO::ready,
                                [this] {
                                    // More stuff...
                                });
                });
}
```

Async initialization chains

```
{
    connect(otherObject, &AsyncInitObject::ready),
                  [this] { /* finalize init here... */ });
    connect(anObject, &AsyncInitObject::ready),
                  otherObject, &AsyncInitObject::init);
}
```

Async initialization chains

```
{
    connect(anObject, &AsyncInitObject::ready,
                  [this, otherObject] {
                      prepare();
                      otherObject->init();
                      connect(otherObject, &ASIO::ready,
                                    [this] {
                                        // More stuff...
                                    }, Qt::QueuedConnection);
                  }, Qt::QueuedConnection);
}
```

Lambdas

Handling UNIX Signals and application lifecycle

main.cpp on steroids

```
static int sighupFd[2];
static int sigtermFd[2];

static void hupSignalHandler(int)
{
    char a = 1;
    ::write(sighupFd[0], &a, sizeof(a));
}

static void termSignalHandler(int)
{
    char a = 1;
    ::write(sigtermFd[0], &a, sizeof(a));
}
```

main.cpp on steroids

```
static int setup_unix_signal_handlers()
{
    struct sigaction hup, term;
    hup.sa_handler = hupSignalHandler;
    sigemptyset(&hup.sa_mask);
    hup.sa_flags = 0;
    hup.sa_flags |= SA_RESTART;
    if (sigaction(SIGHUP, &hup, 0) > 0) {
        return 1;
    }
    term.sa_handler = termSignalHandler;
    sigemptyset(&term.sa_mask);
    term.sa_flags |= SA_RESTART;

    if (sigaction(SIGTERM, &term, 0) > 0) {
        return 2;
    }

    return 0;
}
```

main.cpp on steroids

```
{
    if (::socketpair(AF_UNIX, SOCK_STREAM, 0, sighupFd))
       qFatal("Couldn't create HUP socketpair");

    if (::socketpair(AF_UNIX, SOCK_STREAM, 0, sigtermFd))
       qFatal("Couldn't create TERM socketpair");
    snHup = new QSocketNotifier(sighupFd[1], QSocketNotifier::Read, this);
    connect(snHup, SIGNAL(activated(int)), this, SLOT(handleSigHup()));
    snTerm = new QSocketNotifier(sigtermFd[1], QSocketNotifier::Read, this);
    connect(snTerm, SIGNAL(activated(int)), this, SLOT(handleSigTerm()));

    ...
 }
```

main.cpp on steroids

```
{
    if (::socketpair(AF_UNIX, SOCK_STREAM, 0, sighupFd))
       qFatal("Couldn't create HUP socketpair");

    if (::socketpair(AF_UNIX, SOCK_STREAM, 0, sigtermFd))
       qFatal("Couldn't create TERM socketpair");
    snHup = new QSocketNotifier(sighupFd[1], QSocketNotifier::Read, this);
    connect(snHup, SIGNAL(activated(int)), [this] { /* handle */ });
    snTerm = new QSocketNotifier(sigtermFd[1], QSocketNotifier::Read, this);
    connect(snTerm, SIGNAL(activated(int)), [this] { /* handle */ });


    ...
 }
```

main.cpp on steroids

```
auto startItUp = [&] () {
            core = new Core;
            Operation *op = core->init();
            QObject::connect(op, &Operation::finished, [core, op] {
                if (op->isError()) {
                    qFatal("Initialization of the core failed.”);
                } else {
                    // Do stuff here

                    // Notify system here
                }
            });
    };
```

main.cpp on steroids

```
auto shutItDown = [&] () {
        // Destroy stuff here

        // More stuff here

        QObject::connect(core->lastObject(), &QObject::destroyed,
                         core, &QObject::deleteLater);
  };
```

main.cpp on steroids

```
QSocketNotifier snHup(sighupFd[1], QSocketNotifier::Read);
QObject::connect(&snHup, &QSocketNotifier::activated, [&] () {
        // Handle SIGHUP here
        snHup.setEnabled(false);
        char tmp;
        ::read(sighupFd[1], &tmp, sizeof(tmp));

        qDebug() << "Reloading application...";
        // Destroy & create
        QObject::connect(core, &QObject::destroyed, [&] {
            startItUp();
            snHup.setEnabled(true);
        });
        shutItDown();
});
```

main.cpp on steroids

```
QSocketNotifier snHup(sighupFd[1], QSocketNotifier::Read);
QObject::connect(&snHup, &QSocketNotifier::activated, [&] () {
        // Handle SIGHUP here
});
QSocketNotifier snTerm(sigtermFd[1], QSocketNotifier::Read);
QObject::connect(&snTerm, &QSocketNotifier::activated, [&] () {
        // Handle SIGTERM here
});

if (setup_unix_signal_handlers() != 0) {
    qFatal("Couldn't register UNIX signal handler");
    return -1;
}

// Start the application
startItUp();

return app.exec();
```

Lambdas

Inline management of stateful objects

Lambdas

QDBusPendingCallWatcher

Lambdas

```
QDBusPendingReply<QDBusObjectPath> jobPath;

QDBusPendingCallWatcher *watcher =
                              new QDBusPendingCallWatcher(jobPath);
```

Lambdas

Qt Developer Days 2013

```
auto onJobPathFinished = [this] (QDBusPendingCallWatcher *watcher) {
        QDBusPendingReply<QDBusObjectPath> reply = *watcher;
        if (reply.isError()) {
            setFinishedWithError(reply.error());
            return;
        }
        // Stuff...
    };

QDBusPendingCallWatcher *watcher =
                                        new QDBusPendingCallWatcher(jobPath);

if (watcher->isFinished()) {
        onJobPathFinished(watcher)
} else {
        connect(watcher, &QDBusPendingCallWatcher::finished,
                       onJobPathFinished);
}
```

QTimer

Inline asynchronous code

QTimer

```
QMetaObject::invokeMethod(obj, “myMethod”);

QTimer::singleShot(0, obj, SLOT(myMethod()));
```

QTimer

```
QTimer::singleShot(0, [this] { /* doStuff */ });
```

QTimer

```
{
    int value = computeValue();
    QTimer::singleShot(0, [this, value] { doSthLater(value); });

    // More important things...
}
```

QTimer

```
qAsync([this] {
    qDebug() << “Just a shorter way of ”
                    “doing the same thing.” });
```

QtConcurrent

Inline asynchronous (multithreaded) code

QtConcurrent

```
{
    int value = computeValue();
    QFuture<int>QtConcurrent::run([this, value] -> int {
                    int result = veryHeavyComputation(value);
                    result = result + 42;
                    return result;
    });

    // More important things...
}
```

QtConcurrent

```
{
    int value = computeValue();
    QFuture<int>QtConcurrent::run([this] (int value) -> int {
                    int result = veryHeavyComputation(value);
                    result = result + 42;
                    return result;
    }, value);

    // More important things...
}
```

QTimer

```
{
    int value = computeValue();
    QTimer::singleShot(0, m_objectInOtherThread,
                                    [this, value] { doSthLater(value); });

    // More important things...
}
```

Caveats

Context:
QThread vs. QObject

Caveats

```
{
    QObject *obj = new QObject;
    QThread *thread = new QThread;
    obj->moveToThread(thread);
    thread->start();
    ...

    QTimer::singleShot(0, thread, [] { qDebug() << QThread::currentThread(); });
    QTimer::singleShot(0, obj, [] { qDebug() << QThread::currentThread(); });
}
```

Recap

- Look forward to an even better C++11 experience in Qt 5.2/Qt 5.3
- Use lambdas carefully, and remember context matters
- Upgrade to GCC 4.8.1!

Thank you!

Qt Developer Days 2013

Any questions?

... in case you have some, but you're too shy:

Dario Freddi
dario@ispirata.com