Mágica com Qt's meta-object system

Rodrigo Delduca (@skhaz)

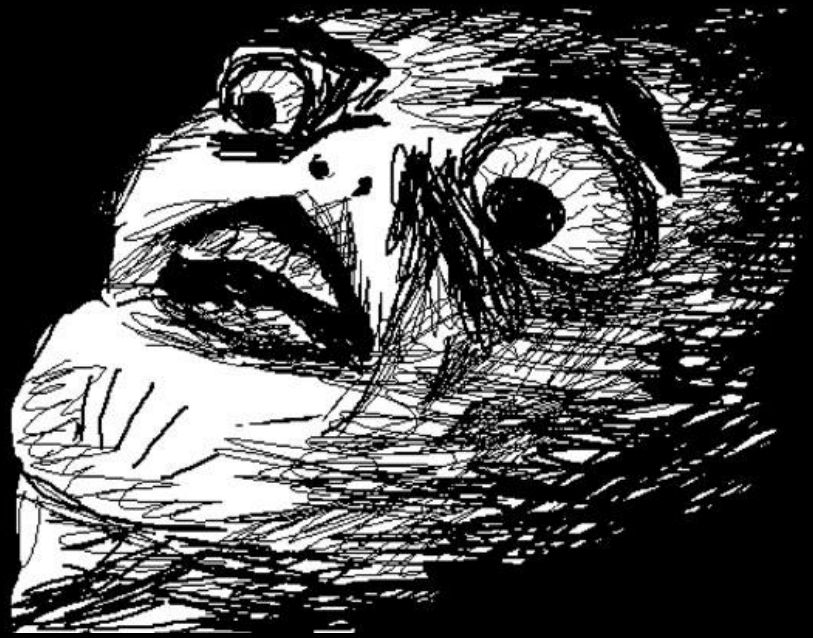
http://nullonerror.org
http://ultratech.software

C/C++ Brasil 2015

```
// FooBar.h
class FooBar : public QObject {
    Q_OBJECT
    public:
        FooBar(QObject *parent = 0);
}

// FooBar.cpp
#include "FooBar.h"
FooBar::FooBar(QObject *parent)
: QObject(parent) {
}
```

```
// FooBar.h
class FooBar : public QObject {
    Q_OBJECT
    public:
        FooBar(QObject *parent = 0);
}
```



QList, QString, QMap... Onde está minha querida STL? Senta que lá vem história...

Qt (/ˈkjuːt/ "cute"): do inglês, "fofinho"

QList, QString, QMap... Onde está minha querida STL? Senta que lá vem história...

Primeira versão em 20/05/1995 (quase 20 anos atrás!)

"The toolkit was called Qt because the letter Q looked appealing in Haavard's Emacs typeface, and "t" was inspired by Xt, the X toolkit" - Wikipedia

TODAS as classes iniciam com a letra Q (~ 1200)

QList, QString, QMap... Onde está minha querida STL? Senta que lá vem história...

Na época os compiladores não suportavam e/ou não existiam recursos avancados, como metaprograming, templates, variadic template, lambda, stl, libsigc++, boost...

Não existia nada como boost.signals2

QList, QString, QMap... Onde está minha querida STL? Senta que lá vem história...

Na época os compiladores não suportavam e/ou não existiam recursos avancados, como metaprograming, templates, variadic template, lambda, stl, libsigc++, boost...

Não existia nada como boost.signals2

A solução foi criar o ***MOC***

MOC (Meta-Object Compiler)

Percorre por todos os headers do

projeto e ao encontrar a macro Q_OBJECT, gera um novo arquivo com o prefixo moc_*.cpp

Suportado por diversas ferramentas de build e IDE como cmake, scons, waf, etc

Can Qt's moc be replaced by C++ reflection?

MOC (Meta-Object Compiler)

E é C++ padrão

Q_OBJECT? De onde vem? O que come? Onde vive?

Macro definido em src/corelib/kernel/qobjectdefs.h

Informa ao MOC (Meta Object Compiler) que que se trata de um Meta Object, gerando código extra (moc_*.cpp)

Obriga que a classe herde de QObject. Toda classe que herde QObject possui os métodos metaObject & staticMetaObject

clang++ -E FooBar.h...

```
class FooBar : public QObject {
    public: template <typename ThisObject> inline void
qt_check_for_QOBJECT_macro(const ThisObject &_q_argument) const { int i =
qYouForgotTheQ_OBJECT_Macro(this, &_q_argument); i = i + 1; } static const
QMetaObject staticMetaObject; virtual const QMetaObject *metaObject()
const; virtual void *qt_metacast(const char *); static inline QString tr(const
char *s, const char *c = 0, int n = -1) { return staticMetaObject.tr(s, c, n); }
static inline QString trUtf8(const char *s, const char *c = 0, int n = -1) { return
staticMetaObject.tr(s, c, n); } virtual int qt_metacall(QMetaObject::Call, int, void
**); private: __attribute__((visibility("hidden"))) static void qt_static_metacall
(QObject *, QMetaObject::Call, int, void **); struct QPrivateSignal {};
    public:
        FooBar(QObject *parent = 0);
}
```

Q_GADGET, a versão *diet* do Q_OBJECT

Definido em src/corelib/kernel/qobjectdefs.h

Não obriga que a classe herde de QObject

"Enxerta" bem menos código

Tem alguns super poderes (exceto os herdados pelo QObject, como por ex. sinais e slots)

Os incríveis super poderes do QMetaObject

- Nome da classe className
- Informações sobre a classe *pai* QMetaObject* superClass()
- Informações sobre os *construtores* int constructorCount() & QMetaMethod constructor(int index)
- Informações sobre os *enumeradores* QMetaEnum enumerator(int index) const, int enumeratorCount() & int indexOfEnumerator(const char * name)
- Informações sobre os métodos, sinais e slots int methodCount() & QMetaMethod method(int index)

Os incríveis super poderes do QMetaObject

- Informações sobre as propriedades int propertyCount() & QMetaProperty property(int index)

- Informações gerais da classe int classInfoCount() & QMetaClassInfo classInfo(int index)

```
class FooBar {
    Q_OBJECT
    Q_CLASSINFO("author", "Rodrigo Delduca")
    Q_CLASSINFO("url", "http://nullonerror.org")
}
```

Os incríveis super poderes do QMetaObject

- Criar uma nova instância QObject* newInstance(args ...)

  qobject_cast<MinhaClasse*>(meta->newInstance());

  NOTA: O construtor de MinhaClasse *precisa ser declarado como Q_INVOKABLE*
- Invocar métodos pelos seus nomes invokeMethod

  QMetaObject::invokeMethod(thread, "quit", Qt::QueuedConnection);

- ...Entre outras coisas :)

Os incríveis super poderes do QMetaObject

- Criar uma nova instância QObject* newInstance(args ...)

  qobject_cast<MeuObject*>(meta->newInstance());

  NOTA: O construtor de MeuObject precisa ser declarado como Q_INVOKABLE

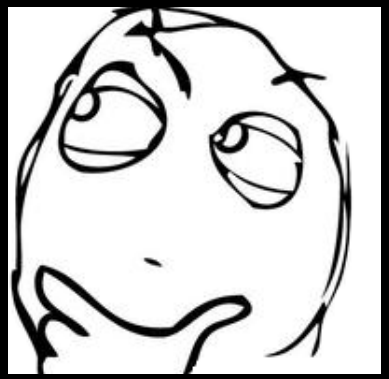
- Invocar métodos pelos seus no

  QMetaObject::invokeMethod(th
QueuedConnection);

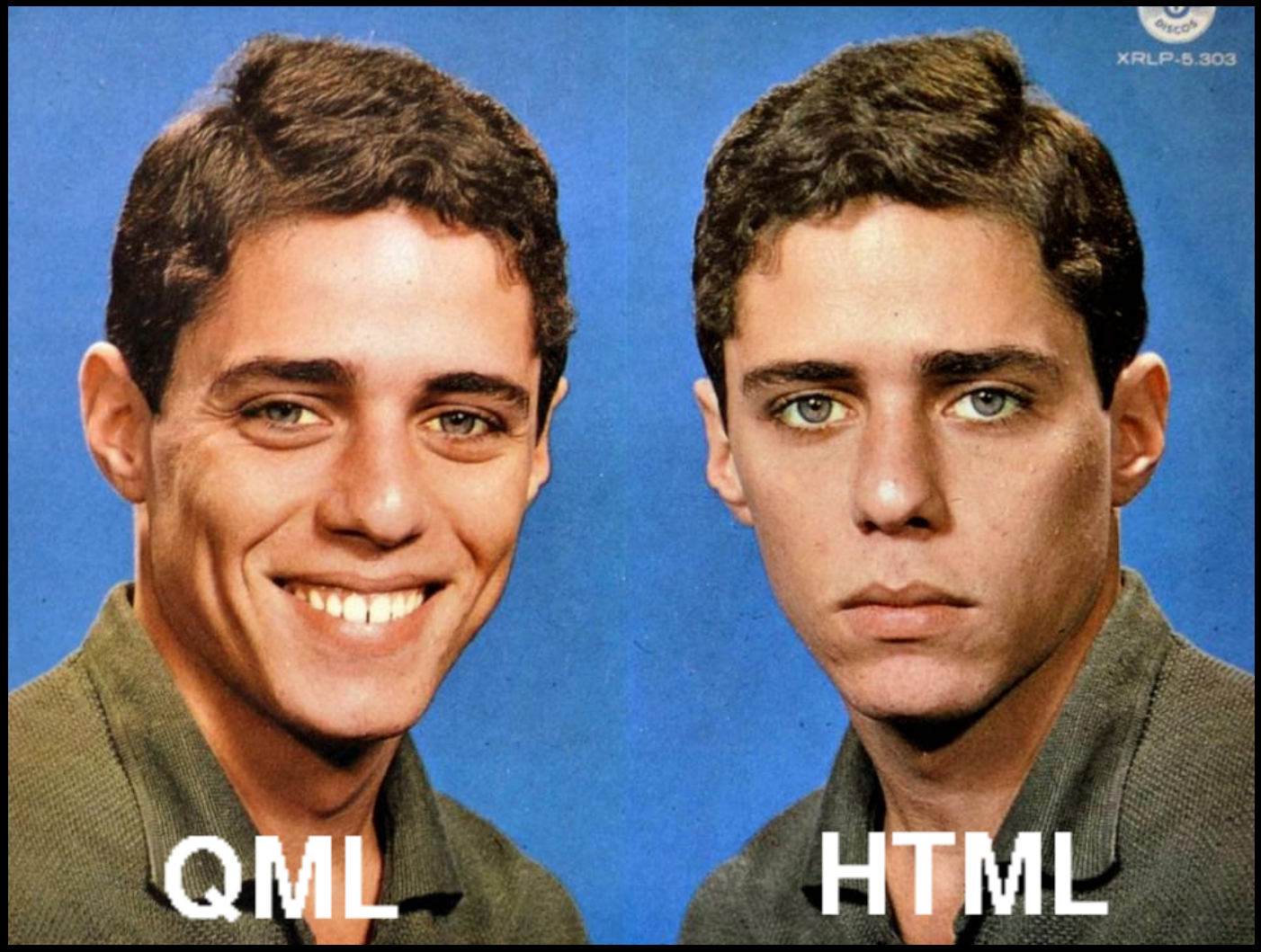
- ...Entre outras coisas :)

Utilidades

- ORM (ex. Qdjango)
- Serialização (ex. QJson QObjectHelper:: qobject2qvariant)
- RPC
- Magia negra...



Aplicações híbridas



Demo

C++ -> QML

C++

```
qmlRegisterType<MinhaClasse>("MeuPacote", 1, 0,
"MinhaClasse");
```

QML

```
import MeuPacote 1.0
MinhaClasse {
  onSignal01: {
     // do something
  }
  words: ['foo', 'bar', 'xpto']
}
```

QML -> C++

QML

```
Button {
  signal clicked()
  objectName: button
}
```

C++

```
QObject *button = root->findChild<QObject *>("button");
if (button)
  QObject::connect(button, SIGNAL(clicked()),
                           this, SLOT(onClicked(QString)));
```

Scorpion wins
Flawless victory
Fatality!

Muito obrigado :)