Qt Promise

Chainable promises for Qt

2017.05.23 - Munich Qt Meetup

Benoit Walter

benoit.walter@meerun.com

Short introduction

A promise has:
- a state (pending, running, fulfilled, failed)
- a result (of a given type) or an error
- usually: an operation (function returning a given type)
- can be chained with another promise

```cpp
auto promise = retrieveData(url);

promise
.then([](const QByteArray &data) {
  // data received
})
.fail([](const PromiseError &error) {
  // Error while retrieving data
});
```

Why promises?

- Make asynchronous operations easier to write
- No code fragmentation (compared to separate "slot" methods or nested callbacks)
- Write async operations in the order of execution
- Error handling
- Safer code with a clear scope and context variables

→ ***Encourage developers to write asynchronous operations without additional complexity***

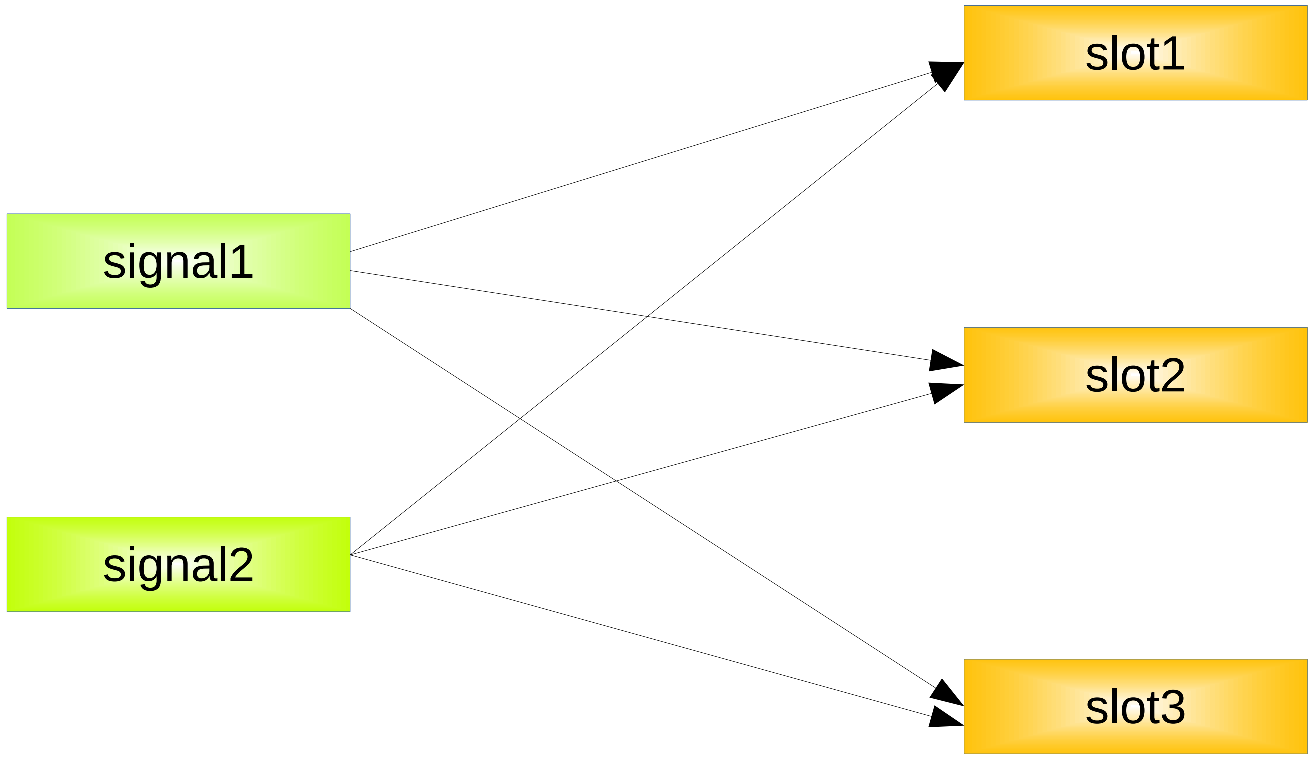
Why Qt Promise?

- Easy to use API for promises
- Integration with Qt event loops
- Limit the lifetime of the promise (e.g. stop when a given QObject has been destroyed)
- Support for QObject connect (signals/slots)
- Support for QThread/QThreadPool
- Compatibility with QtConcurrent/QFuture
- Strongly typed (not variant-based)

→ ***Modern Qt-based code without changing Qt API***

Without promises

Architecture with signals + slot methods



Class members:
State, Values, tmp variables,
QFutureWatcher, QMutex,
QwaitConfition, ...

Without promises (2/4)

Class + private slots

```
class FileDownloader : public QObject {
    Q_OBJECT

  public:
    void asyncOperation();

  signal:
    void done();
    void error(const QString &msg);

  private slots:
    void onDownloadProgress(…);
    void onDownloadError(…);
    void onDownloadFinished();

  private:
    QNetworkAccessManager *m_nam;
    QNetworkReply *m_reply;
};
```

Without promises (3/4)

connect() + nested callbacks

```
…
auto reply = nam->get(QUrl("https://…/api/getid"));

connect(reply, &QNetworkReply::finished, this, [=]() {
  QByteArray id = reply->readAll();
  reply = nam->get(QUrl("…/api/getfile/id"));

  connect(reply, &QNetworkReply::finished, this, [=]() {
     auto data = reply->readAll();
     auto future = QtConcurent::run([=]() {
       …
     });
     QFutureWatcher watcher<QByteArray>;
     watcher.setFuture(future);

     connect(&watcher, &QfutureWatcher::finished,
             this, [=](const QByteArray &result) {
        …
     });
```

Without promises (4/4)

Slot methods:

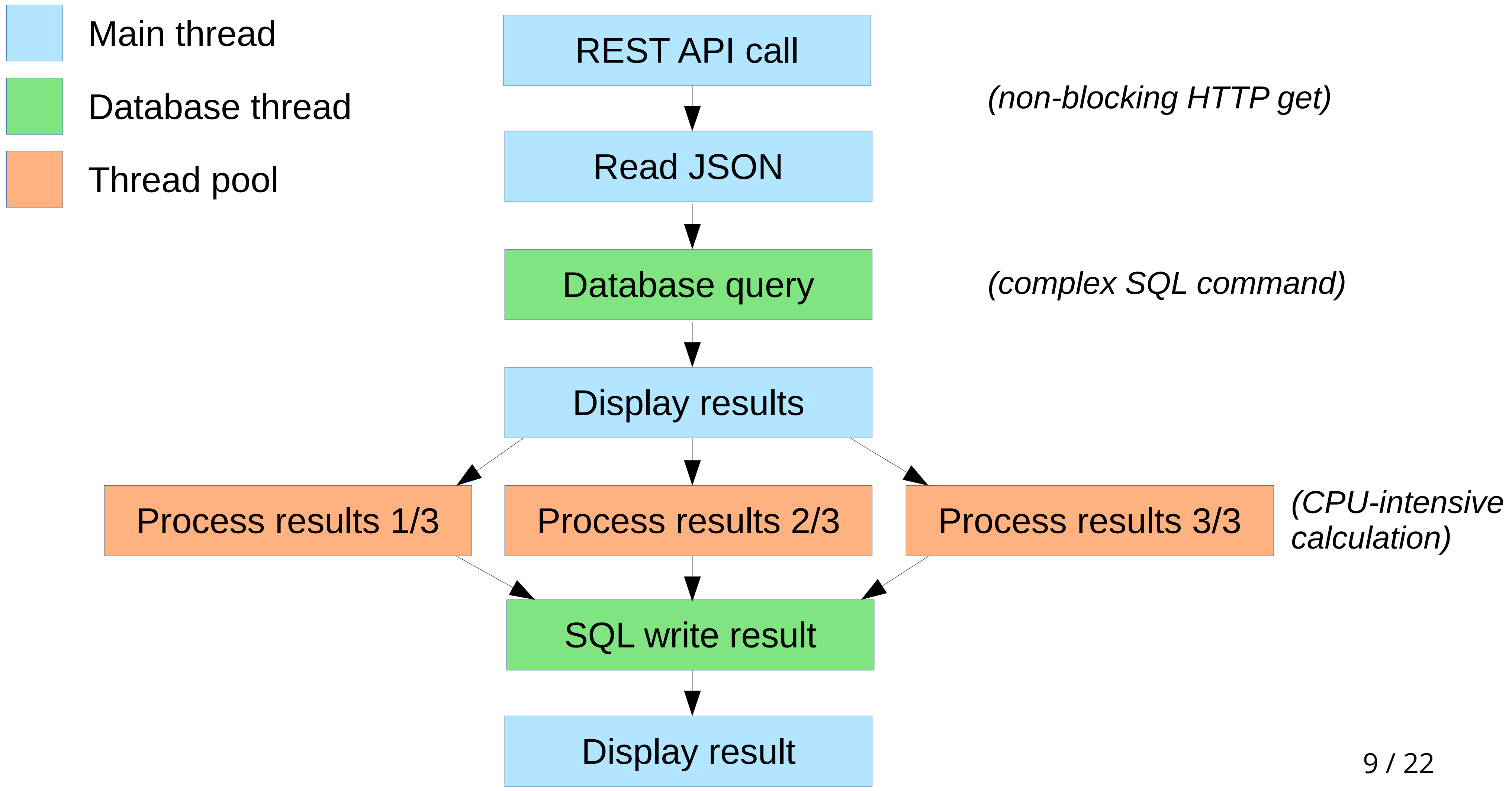
- Fragmented code
- Difficult to run 2 async operations simultaneously:
  → data using class members (scope = class)
  → 1 class instance per operation needed
- Code can hardy reflect the order of operations

Nested callbacks:

- Level of indentation grows with number of calls
- Code difficult to follow
- No easy error handling
- Need to manually disconnect

Multithreading with promise

1 function with promise chain



Using promise

“Flat” promise chain (1/2)

```
auto promise = Promise<void>
.then([=]() {
  nam->get(“https://www.domain.com/api/v1/call”);
  return makeConnectPromise(nam, &QNAM::finished);
})
.then([=](const QNetworkReply *reply) {
  QByteArray json = reply->readAll();
  return json;
})
.then(dbThread, [=](const QByteArray &json) {
  // Read value from DB
  QSqlQuery query(“...”);
  …
  return dbValues;
})
.then([=](const QStringList &values) {
  ...
  return QtConcurrent::mapReduced(…);
});
```

Using promise

“Flat” promise chain (2/2)

```
.then([=](dbThread, const QString &value) {
  // Save value to DB
  QSqlQuery query(“...”);
  …
  return ok ? value : “error”;
})
.then([=](const QString &value) {
  qDebug() << ”Calculated value:“ << value;
})
.fail([=](const PromiseError &error) {
  qWarning() << “Error:” << error.message();
});
```

Creating a promise

Resolved promise with value

```
auto promise1 = Promise<void>();
auto promise2 = Promise<bool>(true);
auto promise3 = Promise<QString>("stringValue");
```

Creating a promise

Chain an existing promise with a lambda

```
auto promise = Promise<void>().then([]() {
  // resolve promise:
  return value;

  // or reject promise:
  throw PromiseError(msg);
});
```

Creating a promise

makePromise + resolve() + reject() (c++14 only!)

```
auto promise = makePromise<T>([](auto resolve, auto reject) {
  // resolve promise...
  resolve(value);

  // ...or reject promise
  reject(PromiseError(msg));
});
```

Creating a promise

Deferred

```
Deferred<T> defer;
auto promise = defer.promise();

// resolve promise...
defer.resolve(value);

// ...or reject promise:
defer.reject(PromiseError(msg));
```

Promise chain

Without error

```
Promise<int> promise = Promise<void>()
.then([]() {
  return “String value”;
})
.then([](const QString &strValue) {
  return 12;
})
.fail([](const PromiseError &error) {
  // not executed because there was no error
})
.finally([]() {
  // always executed
})

promise
.then([](int value) {
  // Executed with value == 12
});
```

Promise chain

With error

```
Promise<int> promise = Promise<void>()
.then([]() {
  throw PromiseError("This is an error");
})
.then([]() {
  // not reached because of the previous error
  return 12;
})
.fail([](const PromiseError &error) {
  // executed with error.message() == “This is an error”
})
.finally([]() {
  // always executed
})

promise
.then([](int value) {
  // Not executed
})
.fail([]() {
  // Executed
});
```

Promise context

QObject instance as context

When giving a QObject as context, we can:

- limit the lifetime of the promise chain: interruption when ctx has been destroyed
- use the context as container for the variables used inside the promise chain
- use the context as parent of QObject instances inside the promise chain
- trigger lambdas in the event loop of the context object thread

Promise context

Example

```
struct ContextObject : QObject {
  int contextVariable = -1;
};

auto ctx = new ContextObject();


auto promise = PromiseContext(ctx)
.then([=]() {
  ctx->contextVariable = 1;
})
.then([]() {
  // Skipped if the context object has been destroyed
})
.fail([](const PromiseError &error) {
  if (error.isContextDestroyed()) {...}
});
```

Connect to QObject signal

makeConnectionPromise()

```
auto timer = new QTimer();
timer->start(3000);

makeConnectionPromise(timer, &QTimer::timeout)
.then([=]() {
  // 3 seconds later...
  return makeConnectionPromise(emitter, &MyClass::intSignal);
})
.then([=](int value) {
  ...
})
.finally([=]() {
  delete timer;
});
```

Connect to Qobject signal

Deferred + connect()

```
auto downloadManager = new DownloadManager();
downloadManager->download("https://www.url.com/file");

Deferred<QByteArray> defer;
defer.connect(emitter, &MyClass::progress, [](double progress) {
  qDebug() << "Progress:" << (int)(progress * 100.0) << "%";
});
defer.connectAndResolve(emitter, &MyClass::downloadSuccessful);
defer.connectAndReject(emitter, &MyClass::downloadError,
                       PromiseError("Download error"));

defer.promise()
.then([](const QByteArray &data) {
  qDebug() << "Downloaded" << data.count() << "bytes!";
})
.fail([](const PromiseError &error) {
  qDebug() << "Failed:" << error.message();
})
.finally([=]() {
  delete downloadedManager;
});
```

Installation and license

- Git Hub: https://github.com/bwalter/qt-promise
- License: Apache v2.0
- 1 single header file: just include in your project
- Contributions (especially bug fixes) welcome
- Unit-tests: see `tests` folder
- Thanks to Ben Lau (Async Future) (https://github.com/benlau/asyncfuture)