Qt Developer Days 2013

Applied Type Erasure in Qt 5

Stephen Kelly
KDAB

Stephen Kelly

- KDAB engineer
- Qt Maintainer (ItemViews, CMake)
- KDE Developer
- Many Qt contributions
  - QVariant
  - QMetaType
- Grantlee - domain specific language
- Boost developer
- CMake developer

Type Erasure

“Process of turning a wide variety of types with a common interface into one type with that same interface.”

Type Erasure

- Hold distinct, unrelated types
- Convert between types
- Store instances in containers
- Copy instances
- Assign instances

Type Erasure

- Interface abstraction
  - QVariant QAbstractItemModel::data();
  - QVariant QVariantAnimation::valueChanged()
- Domain Specific Language
  - QML
  - Grantlee
- Language binding
  - PyQt/PySide
  - RubyQt

Domain Specific Language

```
import QtQuick 2.0
Rectangle {
    color : “lightsteelblue”
    width : 42
    height : 47
}
```

Domain Specific Language

```
Rectangle {
	color : “red”
	color : Qt.rgb(255, 0, 0)
	color : Qt.red
}
```

Text Template System

```
<html>
  <p>Welcome back {{ user.name }}!
  <p>You have {{ user.messages.length }} messages
  {% for message in user.messages %}
    <li>From: {{ message.sender }} : {{ message.content }}
    {% if message.urgent %} <img src=”urgent.png”/> {% endif %}
  {% endfor %}
</html>
```

Text Template System

```
<html>
  <p>Welcome back {{ user.name }}!
  <p>You have {{ user.messages.length }} messages
  {% for message in user.messages %}
    <li>From: {{ message.sender }} : {{ message.content }}
    {% if message.urgent %} <img src=”urgent.png”/> {% endif %}
  {% endfor %}
</html>
```

Requirements

- Type conversion
  - String, numbers
  - Equivalent colors
  - `{% if message.urgent %}`
- Properties
  - `{{ user.name }}`
  - `{{ messages.length }}`
- Containers
  - Sequences
  - Mappings
  - `{% for item in container %}...{% endfor %}`

QVariant

QVariant

Conversion I

- QString QVariant::toString()
  - QVariant(42).toString()
  - QVariant(3.15).toString()
  - QVariant(“Hello, world!”).toString()
- bool QVariant::toBool()
  - QVariant(42).toBool()
  - QVariant(true).toBool()

Properties

```
class User : public QObject
{
    Q_PROPERTY(QString name ...)
    Q_PROPERTY(int numMessages ...)
    Q_OBJECT
    // ...
};

QObject* obj = new User(this);
obj->property(“name”).toString();
obj->property(“numMessages”).toInt();
```

Text Template System

```
<html>
  <p>Welcome back {{ user.name }}!
  <p>You have {{ user.messages.length }} messages
  {% for message in user.messages %}
    <li>From: {{ message.sender }} : {{ message.content }}
    {% if message.urgent %} <img src=”urgent.png”/> {% endif %}
  {% endfor %}
</html>
```

Conversion II

- QObject* QVariant::value<QObject*>()
  - QVariant::fromValue(new QObject)
  - QVariant::fromValue(new QFile)
  - QVariant::fromValue(new User)
  - QVariant::fromValue(QPointer<User>)
  - QVariant::fromValue(QSharedPointer<User>)
  - QVariant::fromValue(QWeakPointer<User>)

Conversion II

```
class User : public QObject
{
  Q_OBJECT
  // ...
};

auto sp = QSharedPointer<User>::Create();
QVariant var = QVariant::fromValue(sp);

// Later:
QObject *obj = var.value<QObject*>()
QString propValue =
obj->property(“some_prop”).toString();
```

Text Template System

```
<html>
  <p>Welcome back {{ user.name }}!
  <p>You have {{ user.messages.length }} messages
  {% for message in user.messages %}
    <li>From: {{ message.sender }} : {{ message.content }}
    {% if message.urgent %} <img src=”urgent.png”/> {% endif %}
  {% endfor %}
</html>
```

Sequential Containers

```
class Message : public QObject
{
    Q_OBJECT
    QPROPERTY(QString content ...)
    QPROPERTY(QString sender ...)
};

class User : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QList<Message*> messages ...)
};
```

Sequential Containers

```
QObject *userObject = ...;
QVariant var = userObject->property(“messages”);

// Can't do this:
QList<Message*> list = var.value<QList<Message*> >();

// Can't do this:
QList<QObject*> list = var.value<QList<QObject*> >();

// Can do this (Qt 5.2)!
QVariantList list = var.value<QVariantList>();
```

Sequential Containers

```
if (var.canConvert<QVariantList>()) {
    auto iter = var.value<QSequentialIterable>();
    foreach(const QVariant &item, iter) {
        // item.toString();
        // item.value<QObject*>();
    }
}
```

Sequential Containers

```
if (var.canConvert<QVariantList>()) {
    auto iter = var.value<QSequentialIterable>();
    for (auto it = iter.begin();
              it != iter.end(); ++it) {
        // it->toString();
        // it->value<QObject*>();
    }
}
```

Sequential Containers

```
if (var.canConvert<QVariantList>()) {
    auto iter = var.value<QSequentialIterable>();
    for (QVariant item : iter) {
        // item.toString();
        // item.value<QObject*>();
    }
}
```

Sequential Containers

- Built-in support for:
  - QList
  - QVector
  - QStack
  - QQueue
  - QSet
  - QLinkedList
  - std::vector
  - std::list

Associative Containers

Qt Developer Days 2013

```
QObject *userObject = ...;
QVariant var = userObject->property("some_mapping");

// Can't do this:
QHash<QString, Message*> mapping =
                   var.value<QHash<QString, Message*> >();

// Can't do this:
QHash<QString, QObject*> mapping =
                   var.value<QHash<QString, QObject*> >();

// Can do this (Qt 5.2)!
QVariantHash mapping = var.value<QVariantHash>();
```

Associative Containers

```
if (var.canConvert<QVariantHash>()) {
    auto iter = var.value<QAssociativeIterable>();
    for (auto it = iter.begin();
              it != iter.end(); ++it) {
        // it.key().toString();
        // it.value().toString();
    }
}
```

Associative Containers

- Built-in support for:
  - QHash
  - QMap
  - std::map

Associative Containers

```
if (var.canConvert<QVariantPair>()) {
    auto pair = var.value<QVariantPair>();
    // pair.first().toString();
    // pair.second().toString();
}
```

- Built-in support for:
  - QPair<T, U>
  - std::pair<T, U>

Text Template System

```
<html>
  <p>Welcome back {{ user.name }}!
  <p>You have {{ user.messages.length }} messages
  {% for message in user.messages %}
    <li>From: {{ message.sender }} : {{ message.content }}
    {% if message.urgent %} <img src="urgent.png"/> {% endif %}
  {% endfor %}
</html>
```

Conversion III

```
Q_DECLARE_SMART_POINTER_METATYPE(std::shared_ptr)

Q_DECLARE_SEQUENTIAL_CONTAINER_METATYPE(
    std::deque)

Q_DECLARE_ASSOCIATIVE_CONTAINER_METATYPE(
    std::unordered_map)
```

Conversion IV

```
struct Roles
{
  QString toString() const;
};
QMetaType::registerConverter(&Roles::toString);

QMetaType::registerConverter(converterFunction);

QMetaType::registerConverter(converterFunctor);
```

Conversion IV

```
#include <QDebug>

struct Roles
{
  bool canDelete() const; bool canCreate() const; bool canAccess() const;

  QString toString() const {
    return “Roles”;
  }
};
Q_DECLARE_METATYPE(Roles)

int main(int argc, char **argv)
{
    Roles r;
    QMetaType::registerConverter(&Roles::toString);
    QVariant v = QVariant::fromValue(r);
    qDebug() << v.toString();
}
```

Summary

- Type erasure based on QVariant
- Qt 5.2 new capabilities
  - Generic QObject* handling
  - Generic smart pointer handling
  - Generic container iteration
  - User extensible
  - User-defined conversions

Q & A

stephen.kelly@kdab.com