SACI::QT::SPINBOX

Usando `coruja::object<T>` como modelos em widgets Qt

30 de Julho de 2019

Ricardo Cosme

Instituto Tecgraf de Desenvolvimento de
Software Técnico-Científico da PUC-Rio
Tecgraf/PUC-Rio

WIDGETS QT

SOLUÇÃO ADHOC #1 - MODELO NÃO OBSERVÁVEL

```
1  struct person_t { double height; };
2
3  class view_t : public QObject {
4    Q_OBJECT
5  public:
6    view_t() : height(&window) {
7      QObject::connect(&height, SIGNAL(valueChanged(double)),
8                       this, SLOT(height_change_slot(double)));
9    }
10   virtual ~view_t() = default;
11   QMainWindow window;
12   QDoubleSpinBox height;
13   coruja::signal<void(double)> on_height_change;
14 public Q_SLOTS:
15   void height_change_slot(double v)
16   { on_height_change(v); }
17 };
```

SOLUÇÃO ADHOC #1 - MODELO NÃO OBSERVÁVEL

```
1  struct person_t { double height; };
2
3  class view_t : public QObject {
4    Q_OBJECT
5  public:
6    view_t() : height(&window) {
7      QObject::connect(&height, SIGNAL(valueChanged(double)),
8                       this, SLOT(height_change_slot(double)));
9    }
10   virtual ~view_t() = default;
11   QMainWindow window;
12   QDoubleSpinBox height;
13   coruja::signal<void(double)> on_height_change;
14 public Q_SLOTS:
15   void height_change_slot(double v)
16   { on_height_change(v); }
17 };
```

SOLUÇÃO ADHOC #1 - MODELO NÃO OBSERVÁVEL

```
1  struct person_t { double height; };
2
3  class view_t : public QObject {
4    Q_OBJECT
5  public:
6    view_t() : height(&window) {
7      QObject::connect(&height, SIGNAL(valueChanged(double)),
8                       this, SLOT(height_change_slot(double)));
9    }
10   virtual ~view_t() = default;
11   QMainWindow window;
12   QDoubleSpinBox height;
13   coruja::signal<void(double)> on_height_change;
14 public Q_SLOTS:
15   void height_change_slot(double v)
16   { on_height_change(v); }
17 };
```

SOLUÇÃO ADHOC #1 - MODELO NÃO OBSERVÁVEL

```
1  struct person_t { double height; };
2
3  class view_t : public QObject {
4    Q_OBJECT
5  public:
6    view_t() : height(&window) {
7      QObject::connect(&height, SIGNAL(valueChanged(double)),
8                       this, SLOT(height_change_slot(double)));
9    }
10   virtual ~view_t() = default;
11   QMainWindow window;
12   QDoubleSpinBox height;
13   coruja::signal<void(double)> on_height_change;
14 public Q_SLOTS:
15   void height_change_slot(double v)
16   { on_height_change(v); }
17 };
```

SOLUÇÃO ADHOC #1 - MODELO NÃO OBSERVÁVEL

```
1  //inicializa view
2  view.height.setValue(person.height);
3
4  //visão atualiza modelo
5  _conn = view.on_height_change.connect(
6    [&](double v){ person.height = v; });
```

SOLUÇÃO ADHOC #1 - MODELO NÃO OBSERVÁVEL

```
1  //inicializa view
2  view.height.setValue(person.height);
3
4  //visão atualiza modelo
5  _conn = view.on_height_change.connect(
6    [&](double v){ person.height = v; });
```

SOLUÇÃO ADHOC #1 - MODELO NÃO OBSERVÁVEL

```
1  //inicializa view
2  view.height.setValue(person.height);
3
4  //visão atualiza modelo
5  _conn = view.on_height_change.connect(
6    [&](double v){ person.height = v; });
```

Problema: múltiplas visões para o modelo

SOLUÇÃO ADHOC #2 - MODELO OBSERVÁVEL

```
1  struct person_t { coruja::object<double> height; };
2
3  //visão atualiza modelo
4  _conn = view.on_height_change.connect(
5    [&](double v){ person.height = v; });
6
7  //inicializa view e modelo atualiza visão
8  conn_ = person.height.for_each(
9    [&](double v){ view.height.setValue(v); });
```

SOLUÇÃO ADHOC #2 - MODELO OBSERVÁVEL

```
1  struct person_t { coruja::object<double> height; };
2
3  //visão atualiza modelo
4  _conn = view.on_height_change.connect(
5    [&](double v){ person.height = v; });
6
7  //inicializa view e modelo atualiza visão
8  conn_ = person.height.for_each(
9    [&](double v){ view.height.setValue(v); });
```

SOLUÇÃO ADHOC #2 - MODELO OBSERVÁVEL

```
1  struct person_t { coruja::object<double> height; };
2
3  //visão atualiza modelo
4  _conn = view.on_height_change.connect(
5    [&](double v){ person.height = v; });
6
7  //inicializa view e modelo atualiza visão
8  conn_ = person.height.for_each(
9    [&](double v){ view.height.setValue(v); });
```

Problema: Ciclo view -> model -> view

SOLUÇÃO ADHOC #2 - MODELO OBSERVÁVEL

```
1  view.height_conn = conn_.get();
2
3  void view_t::height_change_slot(double v) {
4    if(height_conn != coruja::any_connection{}) {
5      auto bc = make_scoped_blocked_connection(height_conn);
6      on_height_change(v);
7    } else on_height_change(v);
8  }
```

SOLUÇÃO ADHOC #2 - MODELO OBSERVÁVEL

```
1  view.height_conn = conn_.get();
2
3  void view_t::height_change_slot(double v) {
4    if(height_conn != coruja::any_connection{}) {
5      auto bc = make_scoped_blocked_connection(height_conn);
6      on_height_change(v);
7    } else on_height_change(v);
8  }
```

SOLUÇÃO ADHOC #2 - MODELO OBSERVÁVEL

```
1  struct person_t { coruja::object<double> height; };
2  class view_t : public QObject {
3    Q_OBJECT
4  public:
5    view_t() : height(&window) {
6      QObject::connect(&height, SIGNAL(valueChanged(double)),
7                       this, SLOT(height_change_slot(double)));
8    }
9    virtual ~view_t() = default;
10   QMainWindow window;
11   QDoubleSpinBox height;
12   coruja::signal<void(double)> on_height_change;
13   coruja::any_connection height_conn;
14 public Q_SLOTS:
15   void height_change_slot(double v) {
16     if(height_conn != coruja::any_connection{}) {
17       auto bc = coruja::make_scoped_blocked_connection(height_conn);
18       on_height_change(v);
19     } else on_height_change(v);
20   }
21 };
22 _conn_0 =  person.height.for_each(
23   [&](double v){ view.height.setValue(v); });
24 view.height_conn = conn_0.get();
25 _conn_1 = view.on_height_change.connect(
26   [&](double v){ person.height = v; });
```

SOLUÇÃO ADHOC #2 - MODELO OBSERVÁVEL

```
1  struct person_t { coruja::object<double> height; };
2  class view_t : public QObject {
3    Q_OBJECT
4  public:
5    view_t() : height(&window) {
6      QObject::connect(&height, SIGNAL(valueChanged(double)),
7                       this, SLOT(height_change_slot(double)));
8    }
9    virtual ~view_t() = default;
10   QMainWindow window;
11   QDoubleSpinBox height;
12   coruja::signal<void(double)> on_height_change;
13   coruja::any_connection height_conn;
14 public Q_SLOTS:
15   void height_change_slot(double v) {
16     if(height_conn != coruja::any_connection{}) {
17       auto bc = coruja::make_scoped_blocked_connection(height_conn);
18       on_height_change(v);
19     } else on_height_change(v);
20   }
21 };
22 _conn_0 =  person.height.for_each(
23   [&](double v){ view.height.setValue(v); });
24 view.height_conn = conn_0.get();
25 _conn_1 = view.on_height_change.connect(
26   [&](double v){ person.height = v; });
```

SOLUÇÃO ADHOC #2 - MODELO OBSERVÁVEL

```
1  struct person_t { coruja::object<double> height; };
2  class view_t : public QObject {
3    Q_OBJECT
4  public:
5    view_t() : height(&window) {
6      QObject::connect(&height, SIGNAL(valueChanged(double)),
7                       this, SLOT(height_change_slot(double)));
8    }
9    virtual ~view_t() = default;
10   QMainWindow window;
11   QDoubleSpinBox height;
12   coruja::signal<void(double)> on_height_change;
13   coruja::any_connection height_conn;
14 public Q_SLOTS:
15   void height_change_slot(double v) {
16     if(height_conn != coruja::any_connection{}) {
17       auto bc = coruja::make_scoped_blocked_connection(height_conn);
18       on_height_change(v);
19     } else on_height_change(v);
20   }
21 };
22 _conn_0 =  person.height.for_each(
23   [&](double v){ view.height.setValue(v); });
24 view.height_conn = conn_0.get();
25 _conn_1 = view.on_height_change.connect(
26   [&](double v){ person.height = v; });
```

SACI::QT::SPINBOX

SOLUÇÃO SACI

```
1  struct person_t { coruja::object<double> height; };
2
3  struct view_t {
4    QMainWindow window;
5  private:
6    QDoubleSpinBox _height;
7  public:
8    view_t(coruja::object<double>& mheight)
9      : _height(&window)
10     , height(mheight, _height)
11   {}
12   saci::qt::spinbox height;
13 };
```

SOLUÇÃO SACI

```
1  struct person_t { coruja::object<double> height; };
2
3  struct view_t {
4    QMainWindow window;
5  private:
6    QDoubleSpinBox _height;
7  public:
8    view_t(coruja::object<double>& mheight)
9      : _height(&window)
10     , height(mheight, _height)
11   {}
12   saci::qt::spinbox height;
13 };
```

SOLUÇÃO SACI

```
1  struct person_t { coruja::object<double> height; };
2
3  struct view_t : window_t {
4      view_t(coruja::object<double>& mheight)
5      : height(mheight, child<QDoubleSpinBox>("height")) {}
6
7      saci::qt::spinbox height;
8  };
```

SOLUÇÃO SACI

```
1  template<typename Model>
2  spinbox::spinbox(Model& model, QDoubleSpinBox& widget)
3
4  const QDoubleSpinBox& spinbox::widget() const noexcept
5  { return *_widget; }
6
7  QDoubleSpinBox& spinbox::widget() noexcept
8  { return *_widget; }
```

SOLUÇÃO SACI

```
1  template<typename Model>
2  spinbox::spinbox(Model& model, QDoubleSpinBox& widget)
3
4  const QDoubleSpinBox& spinbox::widget() const noexcept
5  { return *_widget; }
6
7  QDoubleSpinBox& spinbox::widget() noexcept
8  { return *_widget; }
```

SOLUÇÃO SACI

```
1  template<typename Model>
2  spinbox::spinbox(Model& model, QDoubleSpinBox& widget)
3
4  const QDoubleSpinBox& spinbox::widget() const noexcept
5  { return *_widget; }
6
7  QDoubleSpinBox& spinbox::widget() noexcept
8  { return *_widget; }
```

`spinbox`, `checkbox` e `radio_btn`,

CORUJA::FOR_EACH

Merge de ObservableObject para uma única reação

```
1  auto conn = for_each(draw_person_t{},
2                       person.height,
3                       person.weight);
```

TODO

```
1 template<typename ObservableObjectView>
2 void spinbox::enable(ObservableObjectView&&)
```

TODO

```
1 template<typename ObservableObjectView>
2 void spinbox::enable(ObservableObjectView&&)
```

→ Abstração da GUI

→ Reuso de código ao trocar de engine